NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE DESIGN AND IMPLEMENTATION OF A VISUAL USER INTERFACE FOR A STRUCTURED MODEL MANAGEMENT SYSTEM

by

David D. O'Dell

March 1988

Thesis Advisor: D.R. Dolk

REPORT DOCUMENTATION PAGE

a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited
b DECLASSIFICATION / DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) Code 54	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

1 TITLE (Include Security Classification)
THE DESIGN AND IMPLEMENTATION OF A VISUAL USER INTERFACE FOR A STRUCTURED MODEL MANAGEMENT SYSTEM

2. PERSONAL AUTHOR(S)
O'Dell, David D.

3a. TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1988 March	178

6 SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

7 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Visual Interface; Model Management System; Structured Modeling; Human Factors; Man/Machine Interaction; Graphics Interface

9 ABSTRACT (Continue on reverse if necessary and identify by block number)

In the scheme of an integrated decision support system, model management holds a position comparable with data management. Unfortunately, the development and formalizing of model management techniques historically have lagged far behind data management concepts, although the increased interest in spreadsheets has rekindled an interest in models as productivity enhancing tools. Model management systems offer one way of integrating models into the overall structure of an organizational information resource library. This thesis proposes the design and implementation of a visual interface to one such model management system, based on A.M. Geoffrion's structured modeling paradigm. Our goal is to provide the user with a natural, easy-to-use interface that is, at the same time, powerful enough to extract the full potential from a model management system.

0 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
2a. NAME OF RESPONSIBLE INDIVIDUAL Prof. D.R. Dolk	22b TELEPHONE (Include Area Code) (408) 646-2260 22c. OFFICE SYMBOL Code 54Dk

D FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited.

The Design and Implementation of a Visual User Interface for a Structured Model Management System

by

David D. O'Dell
Major, United States Marine Corps
B.S., Kent State University, 1970

Submitted in partial fulfillment of the requirements for
the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1988

ABSTRACT

In the scheme of an integrated decision support system, model management holds a position comparable with data management. Unfortunately, the development and formalizing of model management techniques historically have lagged far behind data management concepts, although the increased interest in spreadsheets has rekindled an interest in models as productivity enhancing tools. Model management systems offer one way of integrating models into the overall structure of an organizational information resource library. This thesis proposes the design and implementation of a visual interface to one such model management system, based on A. M. Geoffrion's structured modeling paradigm. Our goal is to provide the user with a natural, easy-to-use interface that is, at the same time, powerful enough to extract the full potential from a model management system.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

> The wheel is an extension of the foot, the book is an extension of the eye, clothing, an extension of the skin, electric circuitry, an extension of the central nervous system.
>
> Marshall McLuhan and Quentin Fiore
> The Medium is the Message (1967)

We could easily continue that the computer interface is an extension of the user. To the user, the interface is the program. It is the single direct link that connects him with his application. Regardless of how elaborate the algorithms, how elegant the coding, or how efficient the execution, a poorly designed interface severely detracts from a program's usefulness. At best, the program will be used reluctantly; at worst, it will not be used at all. How the user converses and interacts with the program becomes a prime measure of its effectiveness.

The quality of a program's interface must be a consideration from the beginning to the end of the design process. If a designer fails to take people into account, then his product--be it hardware or software--may well be difficult or impossible to use. As Simpson [Ref. 1] says, human factors[1]

---

1 Human factors is a small but growing discipline which seeks to provide a method for taking into account human strengths and limitations during the design of computer hardware and software [Ref. 1, p. 108]. Historically, human factors has dealt with the physical relationships between man and machine. The development of software pushes even

matter because it is people who must operate the machines [and software] that we produce. Man and machine must work together interactively. The "system" is a combination of both.

Of course, the significance of the user interface varies from application to application. Obviously, the development of any interface represents an expenditure of valuable resources. How much of the available resources one can afford to devote to the interface becomes a legitimate question. Too little and the interface is inadequate. Too much and it becomes an unnecessary aggrandizement. The designer must strive to match the resources expended in developing the interface with the significance of that interface to the application. Simpson [Ref. 1] offers four factors that can provide a measure of the importance of man/machine interaction to software (and concomitantly, interface) design:

(1) the number of people operating the program,

(2) the diversity of the operators' backgrounds,

(3) the complexity of the program, and

(4) the consequences of operator error.

As these factors increase, so should the resources devoted to the development of the user interface. As

---

further--moving into the realm of cognition and touching on exactly how mankind thinks. Because of this cognitive distinction, some researchers prefer to separate the theories of human/computer interaction from the more traditional concepts of human factors.

critical as the interface may be to general software design, an evaluation of the above factors clearly shows that the development of an appropriate user interface assumes added significance in the creation of a highly interactive model management system.

History shows that model-based assistance is used all too infrequently by managers and policy-makers. Often this is the case because available modeling systems are incomprehensible to nonspecialists in management science/operations research (MS/OR) [Ref.2, p.1]. Managers may feel overly dependent on these MS/OR practitioners who more fully understand the underlying concepts of modeling systems. They see this as an erosion of their power. Managers avoid this dependency by avoiding the very modeling systems that could enhance their decision-making capabilities. Modern user interfaces that make the use of modeling systems more intuitive and easier to learn can lead to improved acceptance of the systems themselves. In fact, they may even be a prerequisite for it [Ref. 3, pp. 548-550].

By providing an interface to the modeling environment that is intuitive to the user, he will more likely accept modeling as a powerful, working tool rather than shying away from it as "something better left to those technical types." This thesis proposes one such design. Our goal is to provide the user with a natural, easy-to-use interface that is, at

the same time, powerful enough to extract the full potential from a model management system.

Sections II and III provide a brief overview of both model management and structured modeling, concentrating on those aspects which bear most directly on the user interface. Readers are encouraged to review the specific articles cited in these sections for a more complete treatment of these subjects. Section IV introduces a broad specification of the visual interface concept, touching on various aspects of interface design and interactive computer graphics. Section V provides specifications and design goals of a particular interface, called INTUITION. Section VI describes the prototyping of INTUITION in an IBM PC/MSDOS environment and discusses problems encountered during the design and implementation phases. Finally, Section VII concludes with recommendations for future enhancements to this particular program, as well as proposals for additional research.

II. MODEL MANAGEMENT

Data has long been recognized as a corporate resource to be managed and controlled. Models, too, provide managers with a resource--a decision aid--which, if properly used, can produce a competitive advantage. On the other hand, if used improperly, these same models can lead to disaster.

In the scheme of an integrated decision support system, model management holds a position comparable with data management. Unfortunately, while considerable effort, and consequently considerable progress, has been made in developing and formalizing data management concepts, model management techniques have lagged far behind. [Ref. 2, p.1]

As a result, many organizations have been reluctant in the past to totally embrace model management as a viable information resource. This reluctance has recently diminished with the sky-rocketing popularity of simple modeling devices such as LOTUS 123. These spreadsheets have clearly brought home the unique potential of modeling as an aid to decision-making. But this very popularity, while raising the concept of modeling to the forefront, has come as a two-edged sword. That increased productivity may be gained from effective modeling is a generally accepted premise. And, since productivity determines the relative well-being of an organization, modeling as a tool that increases productivity

is welcomed. On the other hand, the informal, often unstructured, and sometimes unintelligible approach to modeling inherent in the current state of the art, as often as not, overwhelms management. The entire situation is compounded by the management and control problems resulting from the proliferation of spreadsheet models. Model management systems offer some hope of overcoming these obstacles. [Ref. 4]

What is a model management system? As defined by Dolk and Konsynski in the previous reference,

> The MMS [model management system] is to models what the DBMS [data base management system] is to data, i.e., a software system which provides for the creation, manipulation, and access of models.

The key here is "models." Management science and operations research (MS/OR) models are particularly relevant in imposing structure on the decision-making process. Gorry and Krumland [Ref. 5, p. 206] refer to the English Platonist Weldon who suggested that there are troubles which we do not quite know how to handle; there are puzzles whose clear conditions and unique solutions are marvelously elegant; and then there are problems which we invent by finding an appropriate puzzle form to impose upon a trouble. To Gorry and Krumland, MS/OR models are the puzzles imposed on managerial decision-making to solve problems, i.e., to make decisions. The authors continue that the problem(s) in question often fail to dictate the use of any particular model. We may find the

same problem solved in different situations through the use of decidedly different modeling techniques.

In this respect, an MMS allows a decision-maker to access a collection of several different models or modeling techniques, but to do so in a consistent and standard way. Moreover, these models, in the form of an MMS, can be incorporated more readily into an integrated information resource management (IRM) environment, where they can operate in concert with a database management system.

Figure 1 depicts one possible MMS, organized along the lines of a typical DBMS. Readers are referred to Dolk and Konsynski [Ref. 4] for a more complete discussion of model management, in general, and Figure 1, in particular. A brief synopsis follows. Three groups of people are involved in the creation, management, and use of an MMS. The raison d'etre of an MMS is to allow a decision-maker ultimately to solve problems, typically via a model manipulation language of some sort. Just as a DBMS needs to be managed to be effective, an MMS requires equivalent management. This is vested in the model administrator/model bank administrator. In fact, Figure 1 envisions the MA/MBA operating within an information resource management (IRM) environment where model management and data management coexist as sibling functions. These two functions then share a dictionary/ directory system that is more broadly (and appropriately) called an information resource encyclopedia (IRE) to denote a wider array of

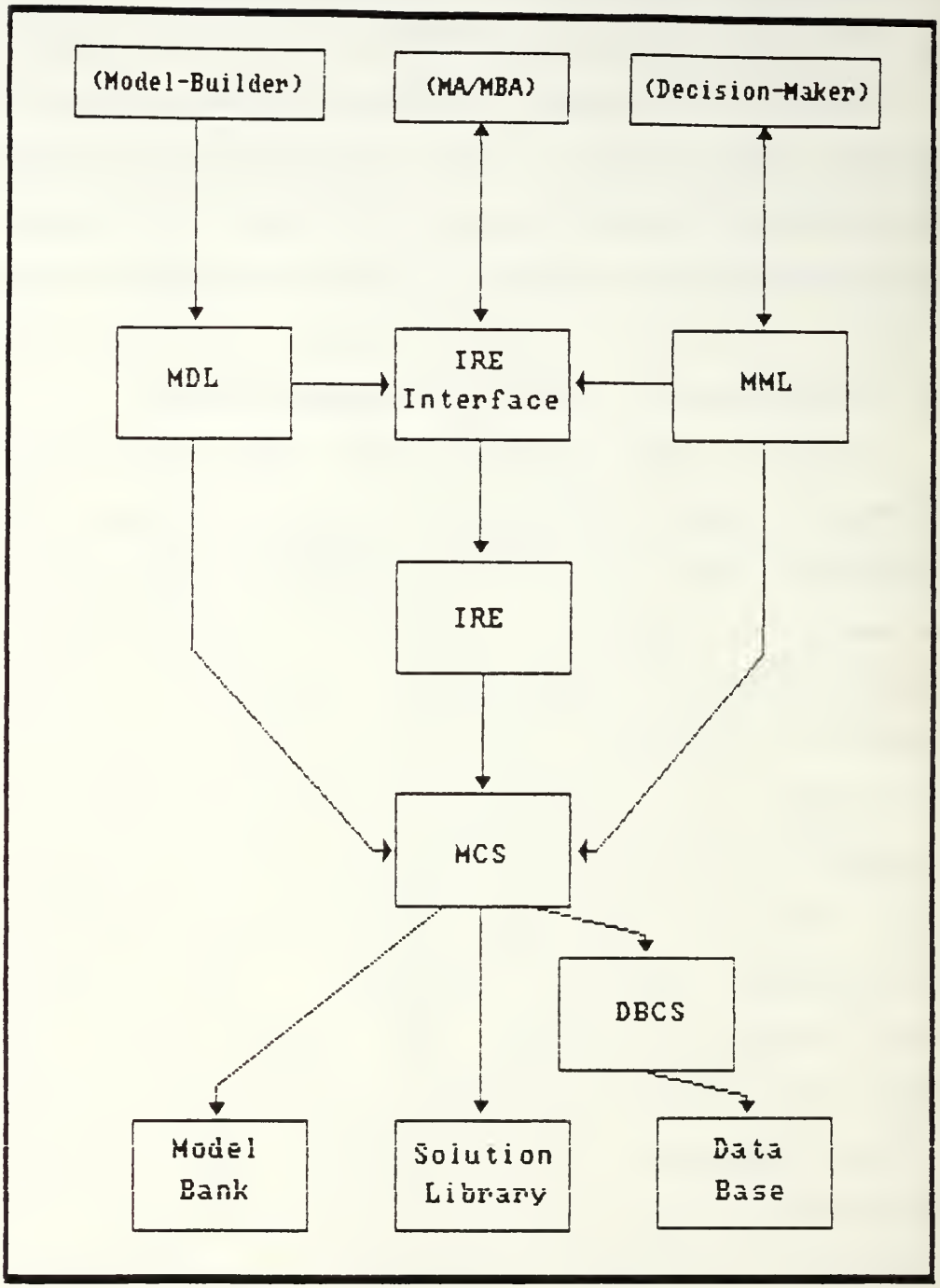

Figure 1: The Components of an MMS
from Dolk and Konsynski [Ref. 1, p.44]

knowledge about a subject--knowledge above and beyond simple definitions. This IRE holds the semantics and structural declarations for both data and models.

Central to the MMS is the model control system (MCS) which manages the physical-level access, storage, and retrieval protocols. The solution library is unique to a MMS and consists of the algorithms necessary for problem solving.

Thus far, we have neglected the left branch of Figure 1, the model-builder. Obviously, the solution library is of no use unless it has something on which and with which "to operate." To this end, the model-builder constructs the model banks and databases which hold the necessary equations and data for the solver to use. He does this through a model description language (MDL). It is this concept of an MDL that is of immediate concern to this thesis. How does the idea, the actual notion, of a model move from man to machine? Dolk and Konsynski [Ref. 4, p. 43] state emphatically,

> Critical to the success of an MMS is a model description language (MDL) which allows users to specify models in a uniform fashion independent of any particular physical implementation....The key advantage of an MDL is that it allows the user to concentrate on model description without worrying about how the model gets solved.

In this same article, the authors suggest, as a logical choice, an algebraic language which facilitates the description of a wide range of mathematical models. This thesis pursues an alternative route--the use of a graphics-based, rather than text-oriented, interface. This requires the adoption of a modeling approach that is:

(1) inherently visual in nature, yet can be adequately described in a text-oriented database, and

(2) of sufficient generality to be applicable across diverse disciplines and varied modeling paradigms.

Structured modeling, developed by A. M. Geoffrion, provides exactly this modeling framework.

III. STRUCTURED MODELING

Structured model theory, as described by Geoffrion [Ref. 3, pp.547-588], provides a structured framework, on par with relational database theory, within which model management requirements can be defined and implemented.

> Structured modeling aims to provide the foundation for a new generation of modeling systems....It also aims to influence how model-based work is carried out using more conventional modeling systems [Ref. 3, p.550].

Whereas previous efforts at developing model management systems have tended to be application specific, Geoffrion's approach is sufficiently general to cross application boundaries. Development of a formalized system around such a paradigm should go a long way in ameliorating the fears and trepidations of management and in advancing the acceptance of model management techniques as standard operating procedures.

Central to the concept of structured modeling is a system of elemental definitions which, in the aggregate, comprise a model. These elements consist of five separate types--primitive entities, compound entities, attributes, functions, and test elements.

(1) A primitive entity element is mathematically undefined. It simply asserts the existence of some thing or concept (e.g., a plant or customer in a transportation system). Every model must contain at least one primitive entity.

(2) A compound entity element represents things or concepts which are defined in terms of previously defined primitive entities and/or other compound entities (e.g., a link in a transportation system consists of a combination of a certain plant and a certain customer).

(3) An attribute element represents properties of things or concepts and is characterized by having a constant, though not necessarily numeric, value within a specified range (e.g., the supply capacity of a plant or the demand of a customer in a transportation system). A variant of the attribute element, called a variable attribute element, differs by having a non-constant value (e.g., the transportation flow between links within a transportation system).

(4) A function element is very similar to the mathematical concept of a function in that it has a unique value within a specified range that is derived by applying a specific rule to the values of other "called" elements (e.g., the total cost associated with all flows in a transportation system).

(5) A test element is simply a special case of the function element. Test elements must evaluate to "TRUE" or "FALSE" and thus provide the capability to perform logical tests on other referenced elements (e.g., is the total flow leaving a plant less than or equal to its supply capacity).

A structured model exists on three levels, each of which provides a varying degree of specificity about the model. The first two levels are organized as acyclic, attributed graphs of distinct elements; the third, hierarchically as a rooted tree.

The base level is the elemental structure. This level comprises "...a nonempty, finite, closed, acyclic collection of elements" [Ref. 6, p. 2-4] which captures the myriad detail associated with a specific instance of the model. For example, in a particular transportation model, there exist

plants in Dallas and Chicago. These plants service customers in Pittsburgh, Atlanta, and Cleveland. Associated with these plants and customers are specific links, flows, and costs that are individually defined in the elemental structure.

A generic structure is defined on the elemental structure by what Geoffrion calls a natural familial grouping of elements. In simple terms, like elements can be aggregated into a single, homogeneous grouping of elements called a genus. Each genus must fulfill the requirement of "generic similarity," that is, each element within the genus must refer to and be referenced by elements from exactly the same genera. From the previous example, individual plants in Dallas and Chicago can be partitioned into the genus, PLANT, while the customers in Pittsburgh, Atlanta, and Cleveland would be partitioned into a single genus, CUSTOMER. Likewise, the various flows, links, and costs would be partitioned into genera, one for each type.

Both the elemental and generic structures are depicted as directed, acyclic graphs called element graphs and genus graphs, respectively. In each case, nodes represent elements (or genera) and arcs represent "calls"[2] of one node by another. Perhaps somewhat counter-intuitively, the head node of each arc is said to "call" the respective tail node.

---

2 A "call" is a definitional reference of one element or genus to another. Calls form calling sequences. These calling sequences are "the cross-references among the various elements of a model [that] are the central focus of structured modeling [Ref.6, p. 2-3].

Figures 2 and 3 provide the element and genus graphs for a typical Hitchcock-Koopmans transportation model.

The third, and highest level, is the modular structure. This is a rooted tree whose terminal nodes have a one to one correlation with the genera of the corresponding generic structure. Non-terminal nodes are defined as modules, with the simplest rooted tree consisting of a single such module--the root module, representing the entire structured model.

Every possible module structure is not allowed. Valid structures must satisfy the requirement of monotone ordering (i.e., must contain no forward referencing). In other words, it must be possible to linearly order all genera in the model in such a manner that no element in a genus ever calls an element in a genus further down the listing. In the vernacular of structure modeling, such a listing is called a modular outline. Valid modular structures are typically depicted as modular trees. Figure 4 shows one possible modular tree and its associated modular outline for the same transportation scenario cited above.

Both generic and modular structures, in the form of genus graphs and modular trees, lend themselves to the graphical display of information. However, Geoffrion has also developed an extensive textual notation to fully describe the generic and modular structures of a model. Called a model schema, it consists of a paragraph for each genus and module

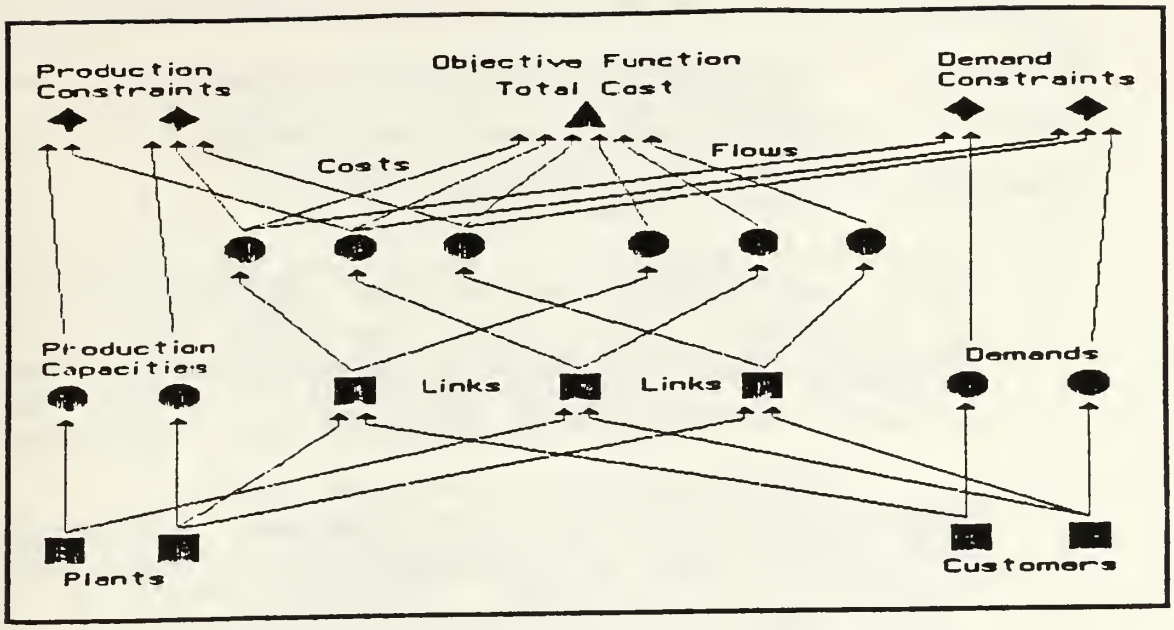

Figure 2: Elemental Structure for a 2 X 2 Transportation Model

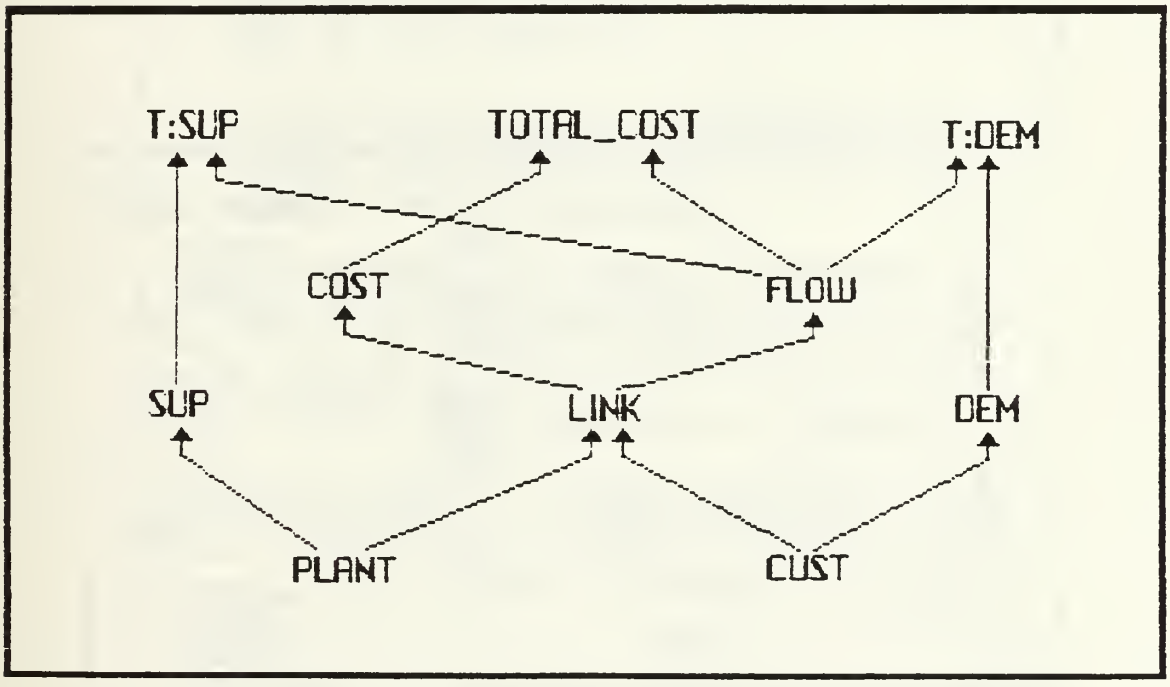

Figure 3: Generic Structure of Transportation Model modified from [Ref. 6, p. 2-4]

in the structure. Figure 5 shows the model schema corresponding to the generic and modular structures given in Figures 3 and 4. Briefly, the syntactic rules for a model schema are:

(1) Paragraphs begin with a unique module or genus name. Names are capitalized. Module names must begin with an ampersand. Genus names must begin with a letter. Names of genera containing multiple elements may be followed by a specific lower case "index" letter, uniquely associated with the genus that introduces it.

(2) Each paragraph contains an informal interpretation part which provides easily readable documentation. Its syntax is unrestricted; however, Geoffrion recommends that it contain an underlined, capitalized and unique key phase which is also capitalized on subsequent use. Unless printed in a different font (e.g., italics), the interpretation should be introduced by some unique character to separate it from the formal part of the paragraph definition. Module paragraphs consist only of a module name and interpretation part.

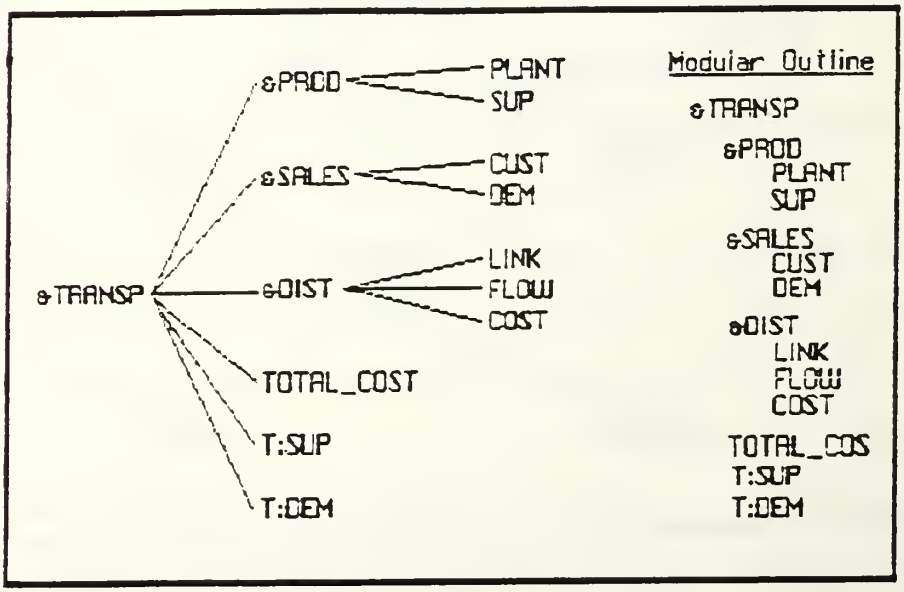

Figure 4: Modular Structure and Outline for Transportation Model from [Ref. 6, p. 2-4]

&PROD There are some PRODUCTION DATA.
PLANTi /pe/ There is a list of PLANTS.
SUP(PLANTi) /a/ <PLANT> : R+ Every PLANT has a SUPPLY CAPACITY measured in tons.

&SALES There are some CUSTOMER DATA.
CUSTj /pe/ There is a list of CUSTOMERS.
DEM(CUSTj) /a/ <CUST> : R+ Every CUSTOMER has a nonnegative DEMAND measured in tons.

&DIST There are some DISTRIBUTION DATA.
LINK(PLANTi,CUSTj) /ce/ Select <PLANT>x<CUST> covering <PLANT>, <CUST> There are transportation LINKS from some PLANTS to some CUSTOMERS.

FLOW(LINKij) /va/ <LINK> : R+ There can be a nonnegative transportation FLOW (in tons) over each LINK.

COST(LINKij) /a/ <LINK> Every LINK has a COST RATE for use in dollars/ton.

TOTAL_COST(COST,FLOW) /f/ 1 ; SUMij(COSTij * FLOWij) There is a TOTAL COST associated with all FLOWS.

T:SUP(FLOWi.,SUPi) /t/ <PLANT> ; SUMj(FLOWij) <= **SUPi** Is the total FLOW leaving a PLANT less than or equal to its SUPPLY CAPACITY? This is called the SUPPLY TEST.

T:DEM(FLOW.j,DEMj) /t/ <CUST> ; SUMi(FLOWij) = DEMj Is the total FLOW arriving at a CUSTOMER exactly equal to its DEMAND? This is called the DEMAND TEST.

Figure 5: Schema Paragraphs for Transportation Model

(3) The formal part of each genus paragraph contains the essential characteristics of the genus:

(a) a type indicator (/pe/,/ce/,/a/,/va/,/f/, or /t/) specifies the element type;

(b) for all non-/pe/ paragraphs, a generic calling sequence (denoted by parentheses) specifies all elements which participate in the definition of a typical element;

(c) an index set statement (denoted by braces) specifies the element population of the genus (omission implies every possible element exists);

(d) for attribute genus paragraphs, a range statement (denoted by a colon) defines the allowable values for the elements of the genus; and

(e) for function and test elements, a generic rule (denoted by a semi-colon) specifies how element values are calculated.

Geoffrion [Ref. 3, p. 550] predicts the potential for wide adoption of structured modeling:

> This kind of definitional system turns out to be widely applicable within model-oriented fields such as MS/OR/DSS (for finance, logistics, marketing, production, and other application areas), information systems, economics, and engineering....structured modeling lays the foundation for a unified theory of model aggregation.

Geoffrion further asserts that such a definitional system is applicable to fields of artificial intelligence, database management, programming language design, and software engineering. It is exactly this generality and capacity for cross-fertilization that makes structured modeling so attractive as the underpinnings of a model management system.

IV. THE VISUAL INTERFACE

Geoffrion [Ref. 6, p. 3-8] describes the idealized interface for a full-scale model management system as providing,

> ...full-screen, fully interactive, mostly command-driven (but with border or hideable menus) access to a file library, which can contain materials pertaining to several models or model schemata.

In view of this description, the question arises, "Why a graphics interface?" This can be answered, perhaps somewhat simplistically, yet nonetheless accurately, by the cliche--a picture is worth a thousand words. The Apple Macintosh, Sun and Apollo workstations, and the forthcoming IBM System 2 series "Presentation Manager" are adequate proof of the acceptance of visual interfaces as productive enhancements to computer technology.

Interactive computer graphics are particularly well suited as a descriptive device for a model management system. As Scott [Ref. 7, p. 77] remarks, "Interactive computer graphics is based on the concept of working with a model described by information stored in the computer." In concept, the interface for a model management system is not very different from the typical "paint" program or computer-aided design (CAD) program.

Computer graphics is equally well suited to the psychological and intellectual requirements of a user interface. Bennett [Ref. 8, p. 54], in amplifying Foley [Ref. 9] states,

> The payoff [from graphics applications] is high because of the bandwidth [capability for rapid interaction with high-resolution, directly relevant pictures] of the communications channel.

Bennett continues,

> Graphics offer potential for decision makers who can benefit from interaction with computer-generated representations but who are repelled by computer-oriented detail.

Of course, as stated in the introduction, if any interface (graphics or otherwise) is to be effective, the needs and capabilities of the user must be considered. Simpson [Ref. 1] recommends six principles for computer interface design:

(1) Provide feedback. Don't keep users in the dark. Every user action, whether correct or incorrect, should elicit some response from the program. Ensure that feedback is immediate, obvious and placed on the screen where the user anticipates it.

(2) Be consistent. While human beings can tolerate ambiguity, it only serves to reduce their effectiveness. Notwithstanding Ralph Waldo Emerson's assertion that consistency is the "hobgoblin of little minds," consistency within software is essential. Establish a set of rules and follow them compulsively. Ensure that similar functions are performed in a like manner throughout the program. This allows the user to learn one part of the program and to apply this knowledge to other portions, making the program much easier to learn and use.

(3) Minimize human memory demands. Simply put, computers have better memories than people. While people do not always remember things exactly, computers do. Reduce the user's need to memorize commands and mnemonics through the use of displayed menus or other screen prompts. Rely on the machine's memory, not the user's.

(4) Keep the program simple. Simplicity must be a conscious design goal. Continuously edit and pare down the program. Provide those functions that are necessary and useful; however, don't encumber the user with unnecessary functions simply because they are possible.

(5) Match the program to the operator's skill level. Evaluate the skill level(s) of expected users and design the program to match these skills. For larger, more complex programs, this may require a complete task analysis to determine what mission a system must perform, what functions are involved, and what tasks are required to perform these functions. Alternatively, for simpler programs, answers to the following questions may suffice:

(a) What will operators be expected to do?

(b) What decisions must they make?

(c) What must they know to make the decisions?

(d) What skill levels will be required?

(6) Sustain operator orientation. Don't allow the user to become "lost" in the program with no way out. Provide signposts to tell him where he is and to guide him back from whence he came. Provide a consistent way to backtrack to a main menu or to an opening screen.

In summary, Simpson (1982, p.116) reduces these six principles to a single idea,

> ...know the needs of your system users. Recognize that they need feedback to avoid confusion, consistency to ease the learning process, minimal strain on memory capacity, simplicity rather than complexity, demands gauged to their skill levels, and constant, clear orientation.

Simpson addresses two additional areas that have special significance to interface design and are thus worthy of mention. These are data entry and screen design. Briefly, he offers these recommendations:

(1) Data Entry

(a) if data are to be entered from a standardized

form, then the screen should look similar to the form in use;

(b) the program should provide a prompt for every data input, including range limits, default values and formats, where possible;

(c) keep input data length to the absolute minimum consistent with the data being entered;

(d) provide feedback by displaying data on the screen as it is entered;

(e) check all data entries for error;

(f) when an error is detected, alert the user, identify the error, and tell the user how to recover;

(g) place all error messages consistently from screen to screen, preferably near the erroneous entry;

(h) allow the user to edit data during the initial data entry, after a group of data have been entered, and subsequent to the data being stored; and

(i) provide "fail-safe" entry mechanisms for data entry that can cause catastrophic failures, e.g. double prompt the user prior to deleting a file.

(2) Screen Design

(a) access screens by paging, not scrolling;

(b) title all screen displays, preferably centered at the top of the screen;

(c) center screen displays, where possible;

(d) allocate specific screen areas for each type or grouping of information (e.g., prompt line, error messages, mode indicators, etc.) and use these areas consistently;

(e) distinctly separate each area of the screen with mechanisms such as blank rows or columns, lines, or color coding;

(f) keep screens simple and uncluttered through the use of "white space;"

(g) follow prevailing conventions--present information from left to right, top to bottom, left justify text and right justify numbers aligning them on the decimal point;

(h) display information in a recognizable order--for example, alphabetically, numerically, or chronologically; and

(i) break up long strings of data into independently recognizable units--for example, showing a telephone number as (408) 555-1234 is much clearer than 4085551234.

V. SPECIFICATION AND SYSTEM DESIGN

Our purpose is to design and implement a visual interface (using established computer interface design principles) for a model management system based on Geoffrion's structured modeling approach. This interface will allow the user:

(1) to interactively enter a graphical representation of a structured model and,

(2) to produce a relational database representation of this model, using the information from the graphic view augmented with other information entered by the user.

We will follow a structured approach to software design. Software engineering methodologies provide the designer with a systematic means of managing and controlling the overwhelming number of tasks involved in software development. To date, no single methodology has caught and held designers' fancies, although within the Federal Government, the classic life cycle or "waterfall" model generally is dictated. However, in many sectors, both prototyping and fourth generation techniques(4GT)[3] are gaining favor.

Prototyping is especially relevant in this instance. In fact, the design of a visual interface fulfills all three of the criteria given by Pressman [Ref. 10, pp. 148-149],

---

3 this term encompasses a wide array of tools that allow a developer to specify software characteristics at a high level and then have the tool automatically generate source code based on these specifications. (Ref. 10, p. 24)

> ...there are circumstances that require the construction of a prototype at the beginning of analysis since the model is the only means through which requirements can be effectively derived. ...In general any application that creates dynamic visual displays, interacts heavily with a human user, or demands algorithms or combinatorial processing that must be developed in an evolutionary fashion is a candidate for prototyping.

Our prototype interface will be designed and implemented in bit-mapped graphics, using a structured programming language, a standardized graphics library, and a relational database management system. The prototype will support monochrome and color (CGA and EGA or equivalent) systems and will self-configure to the hardware used. Graphics objects will be differentiated by shape and color.

The screen will consist of five fixed windows--a status window, an icon window, a command button window (augmented by pop-up menus), a dialog window, and the workspace (drawing) window. User inputs, error messages, and system prompts appear in the dialog window. The user workspace is a free-form drawing area whose size is limited only by available memory. The user sees a four-way scrollable window on this total workspace, into which he interactively enters genus graphs and/or modular trees directly in pictorial form. Commands and node types will be selected from on-screen displays, pop-up menus, or on-screen prompts. The user will have the choice of using a mouse, cursor keys, or function keys to select options.

The interface must be simple, intuitive, and unobtrusive. It should not distract the user from the task at hand, i.e.,

getting his model into the computer. It should support only those tasks necessary to accomplish the program's purpose, as stated herein. Bells and whistles will be minimized. Just because the interface can accomplish a task does not mean that it necessarily should. Every function must be justified on the basis of increased user productivity. Give the user what he needs--nothing more and nothing less.

The interface must be consistent. Similar functions should look the same and work the same throughout the program. The interface must prompt the user for all data entries and must edit all entries. Input must be accepted in a consistent place and manner; output will be displayed likewise.

The interface must be easy to learn and use. It should minimize human memory requirements and the need to reference external documentation. It must provide immediate and consistent feedback for every user action. Errors must elicit meaningful and helpful error messages (i.e., what is wrong and how to correct it). Finally, the interface must provide "signposts" throughout to keep the user oriented and to provide an easy means of returning to the starting point.

The program presupposes that the user understands structured modeling concepts, but makes no further assumptions regarding the user's computer experience. It will be highly structured and modularized to ease transportability across

systems and to allow future extensions to the program. Specifically, it must allow the user to:

(1) add genus elements (nodes) or modular subtrees to the pictograph;

(2) delete genus elements or modular subtrees from the pictograph (including all arcs into and out of the deleted nodes);

(3) rename genus elements or modules on the pictograph;

(4) change the type of genus elements on the pictograph;

(5) add "calls" (directed arcs) between genus elements and modules;

(6) delete existing "calls" from the pictograph;

(7) move genus elements, modules, and subtrees to a new position on the pictograph and adjust all "calls" and spacing accordingly; and

(8) convert specified portions of a genus graph to a modular subtree and automatically redraw the pictograph correctly.

As the user creates a pictorial representation of the model, the program must transform the genus graph and modular structure from the pictograph form to an appropriate relational database representation. Specifically, the following functions will be supported:

(1) extract the genus name, genus type, and calling sequence from the appropriate genus pictograph (extract module name and monotone ordering only for modules);

(2) create a color-coded input screen (based on the structured model paragraph syntax) and allow the user to enter the index set statement, range statement, generic rule, and informal interpretation for a particular genus (or the informal interpretation only for a particular module);

(3) allow the user to modify all paragraph fields except genus name, module name, genus type, and calling

sequence (in order to maintain data integrity, these items can only be modified from the pictograph);

(4) display an appropriate modular outline;

(5) generate a color-coded, scrollable display of the entire model schema based on the appropriate genus and module paragraphs; and

(6) transform the complete model schema into an appropriate relational database representation;

(7) save the pictographs and descriptive paragraphs to disk for subsequent reloading and editing.

The following criteria must be validated when accepting data entry from the user:

(1) Module Paragraph

(a) Module names must be unique, capitalized and prefixed with an "M_"[4].

(b) The informal interpretation must be identified by a unique font or introductory character.

(2) Genus Paragraph

(a) Genus names must be unique, capitalized and begin with an alphabetic character.

(b) Type indicators must be one of the following mnemonics and must be enclosed in back slashes: pe, ce, a, va, f, or t.

(c) Generic calling sequences apply only to non-/pe/ types and must be enclosed in parentheses.

(d) Element indices must be single, lower-case, alphabetic characters and immediately follow the elements that they index.

(e) Index set statements must be enclosed in curly braces.

---

4 Geoffrion designates modules with an ampersand; however, this character has special meaning within the ORACLE Database.

(f) Range statements apply only to /a/ and /va/ types and must be preceded by a colon.

(g) Generic rules apply only to /f/ and /t/ types and must be preceded by a semi-colon.

(h) The informal interpretation must be identified by a unique font or introductory character.

VI. DETAILED DESIGN AND IMPLEMENTATION

The previous section provides preliminary specifications for a visual interface. A prototyping methodology will be employed to further develop and define these general specifications into a system that meets the user's needs. It is a legitimate question at this point to ask why prototyping is used rather than the more traditional life cycle model for software development.

The classical life cycle model is patterned after the conventional engineering cycle. It calls for a sequential series of events that progress systematically from system engineering through analysis, design, coding, testing, and maintenance. Because the output of one step becomes the input to another, this paradigm is frequently called the "waterfall" model. [Ref. 10, p. 20] This process satisfies the need for an organized methodology that establishes appropriate milestones against which to measure progress and at which to obtain approval to continue. Unfortunately, it also requires the capability to fully and unambiguously define all system specifications during the design process. When full specification fails, as it so often does, those requirements that are omitted or misunderstood become the undetected errors that are so costly to correct in later stages.

Prototyping offers an alternative approach that is particularly valuable in developing functional specifications under conditions of uncertainty. Prototyping techniques may either be incorporated into traditional life cycle methods or, in certain circumstances, may replace the traditional model entirely. In either case, the key to prototyping is its iterative nature. The classical model presents the user with reams of text-based specifications and/or reams of diagrammatic representations of those same paper specifications for the entire system. The user cannot possibly determine if the proposed system meets his needs--especially since he very likely doesn't yet know exactly what his requirements truly are. A prototype, on the other hand, attempts to present the user with a realistic view of the eventual system based on what he does currently know about his requirements. This view is then iteratively refined, as uncertainty is reduced, to produce a software product that ultimately reflects the user's true needs. The user is provided with a "specification" that he can experience directly. [Ref. 11, pp. 178-179]

This software project lends itself to prototyping on several counts. Obviously, the prime purpose of any interface is to interact with the user. In a model management system, the level of interaction is particularly high. Prototyping allows this interaction to begin early in the development process and to continue throughout design and

implementation. In addition, this particular case represents one of the first documented efforts at designing a graphics-based model management interface. Developing a comprehensive set of functional specifications for such a first-time system is an unrealistic endeavor. Simply too many uncertainties exist. As explained above, prototyping is ideally suited to developing specifications under conditions of uncertainty. Finally, when working in a graphics environment, it is essential that the user (and developer) see the system as it exists on the computer screen. The screen size and graphics resolution of current computer displays are limiting at best. It is all too easy to develop concepts that work well on paper--an analog medium--but that do not translate at all to the digital screen. Prototyping allows the developer to get the visual aspects of the graphics environment in front of the user early in the process.

The prototype that we propose is called INTUITION. The initial version (1.0) was developed on an IRIS 2400 dedicated graphics workstation. This version was not intended to provide any specific functionality[5], but was used to validate the basic program structure, to develop the initial screen layout, and to determine any unique requirements for

---

5 It epitomized the "throw away" prototype as a mechanism for identifying software requirements. As Brooks' [Ref. 12] writes, "In most projects, the first system built is barely usable....When a new system concept or new technology is used, one has to build a system to throw away, for even the best planning is not so omniscient as to get it right the first time."

including mouse-control as one method of manipulating the interface.

Because of the highly specialized nature of the IRIS workstation, many graphics functions that are trivial on the IRIS system become major considerations on non-dedicated machines. Consequently, INTUITION (Version 1.1) is designed to test the validity of a visual MMS interface in the MSDOS environment using an IBM PC (XT or AT) or compatible. It is programmed in the Lattice C programming language and uses graphic routines from the HALO Graphics Library. Lattice C was chosen because it supports a wide variety of systems (thus easing future portability issues) and, more importantly, because it directly interfaces with the Oracle Relational Database Management System (RDBMS)[6]. The HALO Graphics Library was selected because of its completeness as a graphics language and because of the large number and variety of graphics devices and other peripherals that it supports (again, a portability issue). This makes it possible to produce a bit-mapped graphics display while minimizing machine-specific graphics programming. INTUITION (V 1.1) has been tested with and specifically supports the IBM Color Graphics Adapter (CGA), generic CGA work-alikes, the IBM Enhanced Graphics Adapter (EGA), and the Sigma Designs Color 400 graphics adapter. However, with minimal modifi-

---

6 ORACLE was previously selected as the RDBMS of choice for the particular MMS to which INTUITION interfaces because of its support of Structured Query Language (SQL).

cation, it can be configured to run on any graphics device supported by the HALO Graphics Library. (See Appendix A for a listing of the currently supported devices.)

Appendix B contains structure charts detailing the basic control structure of INTUITION (V 1.1). Both model management and structured modeling are evolving concepts. As such, numerous changes are to be expected in any software package involving them. To remain effective, the software must easily accommodate these changes as new functions are added and old functions are modified or removed. INTUITION accomplishes this in three ways. First, all constants and data types are consolidated and defined in header files where they are readily available if the program needs modification. Secondly, common-use utility functions are gathered into a single library file (util.c). Thirdly, the control structure of INTUITION accommodates a series of "plug-in/pull-out" modules to implement the main functions of the program. These modules, along with the submodules that support them, are essentially self-contained units that can be modified or even completely removed (i.e., pulled-out) without effecting the remainder of the program. Likewise, as new functions are identified, new modules can be developed and easily plugged into the control structure of the program. Not only do these techniques enhance the extendibility and contractibility of the program, but they also significantly ease the portability of the software across computer systems.

The program listing for INTUITION (V 1.1) is provided in Appendix C. Appendix D is a self-contained (i.e., stand-alone) user's manual which provides commands and detailed procedures for using INTUITION.

The program initially creates a display screen that consists of five fixed windows as specified in the previous section. In addition, two smaller windows (one designated ROW and another designated COL) have been included in the screen layout. These are intended to maintain user orientation in the virtual workspace when full-screen scrolling is finally implemented, but are non-functional in Version 1.1. They are included at this point simply to preclude a major screen redesign in the immediate future. Figure 6 provides a mock-up of this main display screen.

The dimensions of each screen area are defined as floating point constants in a header file (intuit.h) and are entered in the program in HALO Graphics world coordinate mode. This facility has become a standard in languages that support graphics programming. It allows us to enter all measurements in real world units[7] as opposed to dealing strictly in artificial screen coordinates. This significantly eases the process of screen layout and object design.

---

7 In this case, measurements are given in inches and fractions thereof. Specifically, the screen is laid out in a rectangular grid that measures eight inches horizontally by five inches vertically. These proportions reflect the actual aspect ratio of the computer screen reasonably well.

Unless otherwise noted, all measurements in the program are given in world coordinates.

Each of the six icons representing a modeling element is constructed within a rectangle one inch long by eight/tenths inch high. The individual geometric shape fits in a six/tenths inch square centered inside this rectangle. The shapes themselves have no specific significance to structured modeling. They were selected primarily to provide a clear visual differentiation of elemental types. However, the seventh icon representing the structured model concept of a module is approximately twice the size of the other elemental icons to visually show that it fulfills a different function within structured modeling. Each icon is completely defined within a separate module (in the file, icon.c). Consequently,

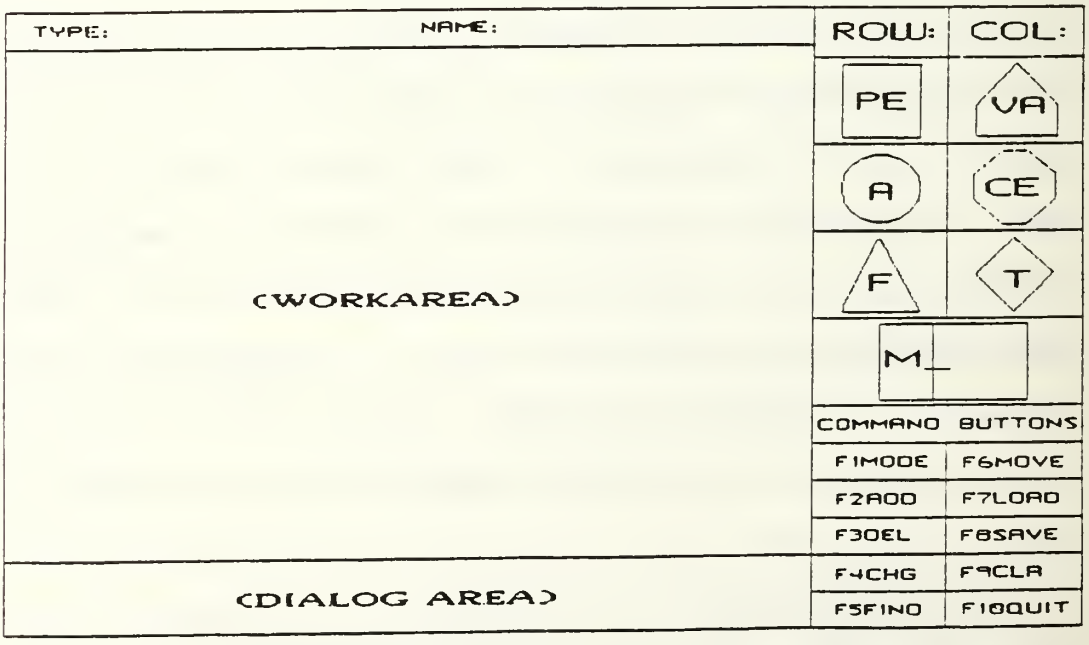

Figure 6: INTUITION (V 1.1) Main Screen

an icon can be reproduced at any point in the program by simply calling its respective module. This also makes it easy to redesign or modify the icons during the prototyping process to better align them with the user's perceptions of what they should be.

Error messages and various prompts are presented to the user in the dialog area at the bottom of the display screen. However, menu choices (other that the initial command button selections) are made from pop-up menus that overlay portions of the workspace area. This necessitates a series of steps each time a menu is displayed:

(1) saving a portion of the workspace to a temporary array,

(2) drawing the pop-up menu,

(3) reading the user's selection,

(4) erasing the menu, and

(5) restoring the screen display underneath.

Moving large portions of the screen display to and from memory is typically a slow process often requiring machine-level programming to make it practical. Fortunately, the HALO Graphics Library provides two functions, MOVETO and MOVEFROM, that greatly simplify this task and are sufficiently fast for our purposes.

The workspace area utilizes another feature of the HALO Graphics Library, the SETVIEWPOINT function. This allows us to set one world coordinate system for the entire screen and a separate world coordinate system for just the workspace

area. This technique allows us to reduce the size of the icons in the workspace to approximately three-fourths of their menu size, thus allowing us to produce somewhat larger models in the workspace area.

The workspace itself consists of 56 cells arranged in a matrix of eight columns by seven rows. Figure 7 shows the layout of a typical cell. The active cell is designated by a box cursor that delineates the cell boundaries. This cursor, as well as the edge cursor, discussed later, are created with

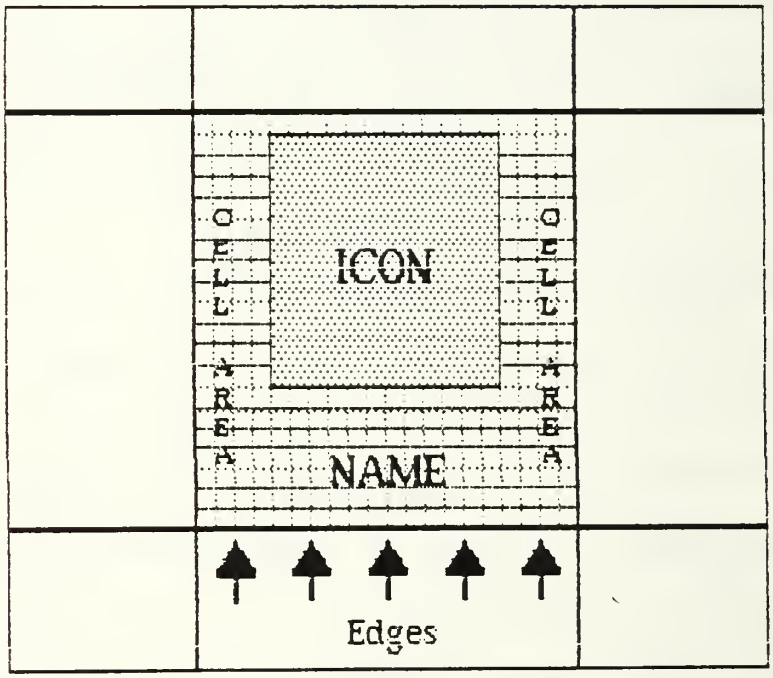

Figure 7: Workspace Cell Layout

the HALO Graphics rubber band functions. This is a set of functions that allow us the easily animate objects by automatically erasing and redrawing their image as their screen coordinates change. These functions use an exclusive

or (XOR) operation to draw and erase the cursor without destroying the underlying screen image.

Selected icons and their associated names are displayed inside the cell boundaries. These cells can then be connected by arcs (edges). INTUITION (V 1.1) currently restricts the numbers of edges (i.e., calls) emanating from a single cell to five or less. Otherwise, every effort was made to allow the user near total freedom in positioning the edges so that the program does not unnecessarily dictate his view of the model. This presents several problems.

First, edges may be drawn at any angle, so lines approaching the horizontal or vertical show varying degrees of aliasing[8]. This can make it difficult for the eye to follow the edge, particularly if lines happen to intersect, as they well may in a complex model. The only solution at this point is to use the maximum resolution possible to reduce the "jaggies," although in the future, various anti-aliasing techniques should be explored to improve the display quality.

The second problem comes from the fact that the edges must be directed arcs. On paper, this is accomplished quite simply by placing an "arrow" on the calling end of the arc. It becomes much more difficult on the computer screen. Since the individual screen pixels are rectangular, not square, the

---

8 This is the stair-step effect commonly seen when circles and lines are drawn on a computer screen.

arrows on non-horizontal or non-vertical edges are severely distorted. To solve this problem, we chose to draw these arcs with short vertical sections that contain the arrow itself (see Figure 7).

Finally, if an edge is simply drawn as straight lines from a beginning point to an ending point, the possibility exists that it will inadvertently pass through an icon. To preclude this from happening, we allow the user to change the direction of an edge (up to three turns in V 1.1) by simply moving the edge cursor to a desired location, pressing a key, and continuing the edge from that point. However, this means that we have to save, not only the beginning and ending coordinates of each edge, but also the coordinates of each turn point. This significantly increases the amount of graphics data that the program must track in order to manipulate the image (e.g. to erase an arc that is incorrectly placed).

The HALO Graphics Kernal contains functions for allocating computer memory and for rapidly saving and recalling the entire graphics screen to and from this allocated block of memory. This technique is used to create a second screen for data entry of the non-graphical portions of the structured model (see Figure 8). The user can toggle back and forth between the two displays at any time without loss of data.

To complete the process of model creation, the graphical representation must be transformed into a database format and stored in the ORACLE RDBMS. INTUITION interfaces with the database through ORACLE's programmatic interface called Pro*C. This enables INTUITION to actually read and write data to ORACLE, using a high-level query language: Structured Query Language, or SQL (pronounced "sequel").

Data is stored in ORACLE in two tables. The first of these is the ENTITY table. It contains the data necessary to describe the specific characteristics of each model node. The second is the RELSHIP table which holds the data that describes the relationship between nodes. In the case of genus graphs, this is the calling sequence data. For modular trees, it reflects what subtrees are contained in the module,

MODEL NAME:		Record NUMBER:
Name:	Description:	
Type:	Last Modified:	Number Mods:
Date Added:		
Index:	Generic Range:	
Index Statement:		
Generic Rule:		
Comments:		
Relationship Type:	Edge 1:	called by
	Edge 2:	called by
	Edge 3:	called by
Relative Position:	Edge 4:	called by
	Edge 5:	called by
[+] Next [-] Previous [e] Edit [f] Find [s] Save [r] Return		

Figure 8: INTUITION (V 1.1) Data Entry Screen

i.e., the parent-child relationships. Refer to Figure 9 for the specific contents of each table.

In the ENTITY table, ENAME and ETYPE hold the entity name and entity type. For structured models, these refer to the specific genus or modular name and the appropriate element type (pe, ce, a, va, f, t, or m). DNAME is a free-form descriptive variant of the entity name. DATE_ADDED is the date that the record was created and LAST_MOD is the date that the record was last modified. NMODS is the number of modifications that have been made to the record. IDX, IDX_STMT, GRANGE, GRULE are the index, index set statement, generic range statement, and generic rule, respectively, from the structured model schema paragraph (see Figure 5). COMMENTS equates to the structured modeling concept of an informal interpretation.

In the RELSHIP table, RTYPE is the relationship type. This is either "CALLS" for a generic calling sequence or "CONTAINZ" for a modular subtree. E1NAME and E1TYPE reflect the calling element and type in a genus graph or the parent node and type in a modular graph. Likewise, E2NAME and E2TYPE are the called element and type in a genus graph or the child node and type in a modular graph. ACC_METH (access method) and FREQ (frequency) are not used in this program and are always null filled. REL_POS is used only in modular structures and contains the positional number of the node in a monotone ordering.

Table Name = 'ENTITY'

Col. No.	Col. Name	Col. Type	Width
1	ENAME	CHAR	12
2	ETYPE	CHAR	8
3	DNAME	CHAR	30
4	DATE_ADDED	DATE	7
5	LAST_MOD	DATE	7
6	NMODS	NUMBER	5
7	IDX	CHAR	4
8	IDX_STMT	CHAR	100
9	GRANGE	CHAR	20
10	GRULE	CHAR	100
11	COMMENTS	CHAR	100

Table Name = 'RELSHIP'

Col. No.	Col. Name	Col. Type	Width
1	RTYPE	CHAR	8
2	E1NAME	CHAR	12
3	E1TYPE	CHAR	12
4	E2NAME	CHAR	12
5	E2TYPE	CHAR	12
6	ACC_METH	CHAR	12
7	FREQ	NUMBER	6
8	REL_POS	NUMBER	6

Figure 9: ORACLE RDBMS Tables

It is obvious that ENTITY and RELSHIP are general purpose tables within the context of the model management system. As such, they contain data fields that are used for a variety of purposes. Consequently, not every item in each table is necessary to describe a given modeling element or relationship. For example, a primitive entity requires only ENAME, DNAME, DATE_ADDED, LAST_MOD, NMODS, and COMMENTS. The remaining fields in the ENTITY table would simply contain null values. On the other hand, a test element or a function element requires values for IDX_STMT and GRULE, in addition to those necessary for a primitive entity. A similar situation holds for the RELSHIP table. A calling sequence requires only E1NAME, E1TYPE, E2NAME, and E2TYPE (the calling and called nodes). However, a modular relationship must also include REL_POS to denote the positional relationship of child to parent in the monotone ordering. All remaining fields are simply null placeholders. This capability to design a general purpose table and then create alternative **views** or ways of looking at the same data is one of the distinct advantages of the ORACLE RDBMS and SQL. The views become **virtual tables** through which you can see the data that is stored in **real tables**. While the views themselves contain no data of their own, they can be operated on just as if they were the real tables. According to the ORACLE documentation, this has the advantage of simplifying data access, of providing data independence, and of providing data privacy.

Version 1.1 of INTUITION falls short of providing the total functionality specified in the previous section. These shortcomings, with recommendations for their inclusion in future versions of the program, are discussed in Section VII under future enhancements to the system.

However, INTUITION (V 1.1) has shown that the implementation of a visual interface for a model management system is feasible in a standard microcomputer environment; but, it is not a trivial undertaking. The primary difficulties stem from the open ended nature of the thinking, and hence the modeling, processes. Different people approach the same problem in a variety of ways. In a computer environment, the hardware and software should transparently assist the modeler in depicting his mental image of the model. It should not unduly restrict or inhibit his thought processes. The program should accommodate him. He should not have to bend his will to the machine.

It is unfortunate for the programmer that every user is different. In general, software cannot be designed for a single individual user, but must satisfy many. It is these differences in users, coupled with the need for free-form entry of a model concept, that make this type of programming so difficult to do well.

Nevertheless, the development of a visual interface poses no unsolvable problems, even though the IBM PC/MSDOS (and

compatibles) environment is perhaps not ideal[9] from a programming perspective. Two areas deserve particular comment.

The first of these is screen resolution. The typical microcomputer display device is a standard cathode ray tube (CRT). It generally uses raster-scan technology to "paint" a picture on the screen by means of an electronic beam. This picture is created as a set of points (called picture elements or pixels) as the electron beam scans from left to right and top to bottom. The number of pixels that can be displayed without overlap is called the CRT's resolution.

Obtaining adequate resolution in the MSDOS environment presents some concerns. Our initial specifications required the program to support both CGA and EGA type graphics adapters and to support color coding of various icons and paragraph schema. It quickly became apparent that the maximum CGA color resolution (320x200 pixels) was severely inadequate. For example, circles smaller than the diameter of a quarter loose their "roundness." Diagonal lines exhibited excessive aliasing, making them unsuitable for drawing directed arcs between nodes.

The coup de grace was the text display. Text is required to label element names within the drawing area. Even in graphics mode, CGA text is displayed at 40 characters per

---

[9] More will be said about the "ideal" environment in Section VIII, under future enhancements.

line. Text this large severely limits the number of nodes that can be displayed simultaneously on the screen. This defeats one of the primary advantages of a visual interface--the ability to see the nodes and arcs in context. The disadvantages of the CGA color mode heavily outweigh the advantage of having four colors available for programming. We decided to limit CGA displays to black and white in order to use the higher resolution (640x200) available in the monochrome display mode. Even then the size of the text display is limiting because text is displayed twice as high as wide. Realistically, EGA (640x385) or better resolution is desirable for any display and is essential for color displays.

The second area of concern is computer memory. Graphics programming, in general, is memory intensive. In addition, this program uses pop-up menus which require that large sections of the screen temporarily be saved to memory when a menu is displayed so that the screen underneath can be restored when the menu is erased. The higher the resolution, the more memory that is required to store the screen information.

Compounding the fact that the graphics themselves use substantial memory, the program needs to accommodate reasonable sized models (i.e., up to several hundred nodes and arcs). The data structures necessary to hold both the graphics data and modeling data for these elements is significant. Finally, the program must share memory with the

Oracle RDBMS which has considerable memory requirements of its own.

The outgrowth of this is that expanded memory, beyond 640K, is necessary for effective use of the system. How much expanded memory is dependent essentially on what size models will be created and what resolution screen is desired. The availability of 640K or less, however, necessarily restricts the user to CGA monochrome and relative small models.

VII. FUTURE ENHANCEMENTS AND ADDITIONAL RESEARCH

Several utilities and functions should be included in the next revision of INTUITION. These enhancements are discussed below.

Scrolling. INTUITION (V 1.1) only supports a limited workspace of 56 cells. While sufficient to test all program functions, this workspace is insufficient to hold any but the smallest models. To be effective, the next version of INTUITION should support a four-way, scrollable "virtual" workspace. The program should determine the actual maximum size of this workspace automatically from the memory available on a given computer and should configure itself accordingly.

Saving and Loading Models. It is not unreasonable to expect the user to enter a model (particularly a large model) in more than a single session. This means that the program must support some method of saving the partial model to disk and subsequently reloading it, without the loss of information or spatial orientation. INTUITION (V 1.1) only supports a limited means of doing this. Basically, the entire graphics screen is saved to disk as an image file. The necessary text information required to describe the model is saved to another file. This method will not work when the workspace is expanded by scrolling. The next version of

INTUITION should support the saving and reloading of all models, regardless of size.

Printing Model Graphs and Model Schema. INTUITION (V 1.1) supports no hardcopy devices, although several printer and plotter routines are included with the HALO Graphics Library. Printing a model schema presents few problems since it is essentially text-based data. Printing the model graph is more difficult because it involves output to a graphics printer or plotter. The HALO Graphics Library eases these problems somewhat by providing driver programs for several common printers and plotters. In addition, the HALO library includes a special graphics device driver called the Virtual Raster Interface. This device driver is unique in that it does not control a specific board. Rather, it creates the model of a display device in the IBM PC user memory space. Such a virtual display device permits the user to create a display of any arbitrary resolution and then output the display to a hard copy device. It is thus possible to produce printouts of much higher resolution than the display device actually installed in the computer. The possible advantages of producing hardcopy with this device driver should be explored in the next version of INTUITION.

Testing. Pressman [Ref. 10, p. 467,470], in describing testing, states,

> Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and coding....The increasing visibility of software as a system element and the attendant "costs"

> associated with a software failure are motivating forces for well-planned, thorough testing. It is not unusual for a software development organization to expend 40 percent of the total project on testing....The design of tests for software and other engineered products can be as challenging as the initial design of the product itself.

Only limited and informal testing of INTUITION has been done. This consisted primarily of "white box" testing required during the coding process. A detailed test plan should be developed and appropriate test cases devised to more fully test the program. This should include "white box" testing to ensure that all independent paths within each function have been exercised at least once, that all logical decisions have been tested on both the true and false sides, that all loops have been tested at their boundaries and within their operational ranges, and that the internal data structures are valid. "Black box" testing should also be conducted to ensure that the program meets all functional requirements. Specifically, it should determine if there are any incorrect or missing functions, if there are any interface errors between functions, if there are errors in data structures or external database access, if there are performance errors, and if there are any initialization or termination errors. [Ref. 10, pp. 472, 484]

With regard to further research, two areas have surfaced. These topics are briefly introduced below.

Re-creation of the Graphic Representation from the Database Representation. Concurrently with this thesis, Wyant [Ref. 13] has designed and implemented a program that

uses the database representation of a structured model to redisplay a pictorial view of the genus graph. In a sense, his thesis and this one are opposite sides of the same coin. While this thesis concentrates on entering a structured model into the database using a visual representation of the genus graph, Wyant's does just the opposite. Both functions are necessary parts of a complete model management interface.

In fact, both programs could be combined into an integrated system. This could provide substantial savings in program overhead. No major problems are anticipated in merging the two. Both are written in the Lattice C programming language, use the HALO Graphics Library, run in the MSDOS environment, use similar icon images for display, and access the same database structures within the Oracle RDBMS. Minor problems, such as adopting a common screen display and eliminating duplicate functions, should be easily resolved. One major concern does stand out, however. When a model is reloaded from the database representation, spatial relationships set up between nodes and arcs when the model was originally designed will be lost. This could cause some disorientation for the user, particularly if he expects to see "his" model re-displayed. Wyant proposes to solutions to this problem:

(1) expand the database to include the necessary graphical data, so that the graph can be re-displayed as originally drawn; or

(2) reformat the user's representation (following Wyant's algorithms) as the user enters the model data.

Moving Genera/Subtrees within the Pictorial Representation. One of the advantages of a computer interface is the ability to quickly and easily modify the model representation to reflect how the user currently views or thinks about the model relationships. To do so, the user must be able to specify a segment of the model pictograph--either a group of genera or a modular subtree--and, once specified, to move this grouping to any other location in the pictograph. This presents major problems that are, in many respects, similar to those faced by Wyant in re-creating the model representation from the database description.

Specifically, the entire pictograph must be realigned to open up space for the group at the new location and to close up the vacated space at the old location. Even more difficult, all of the edge relationships between nodes must be maintained and automatically adjusted for the new location (without intersecting any other icons). Essentially, this requires restructuring of the entire model. In an interactive environment, this must be accomplished in a reasonably short time. It is a potentially significant problem that must be resolved in future versions, quite possibly through employment of artificial intelligence techniques. Alternatively, the algorithms for displaying the model that were developed by Wyant could possibly be used to accomplish this function.

Other Operating Environments[10]. As stated earlier, while the development of a visual interface is certainly feasible under MSDOS, it is not the optimum programming environment for a graphics based system. Several alternatives exist. In particular, recent advances in computer technology provide some very attractive approaches to explore.

As indicated in a previous section, the original version of INTUITION was implemented on a dedicated IRIS Graphics workstation. With its ability to link to the UNIX environment, the IRIS offers an excellent high-end solution. It provides all of the necessary graphics functions and is unique in offering the capability of viewing graphics objects in three dimensions. Whether or not there is any advantage in being able to visualize a structured model (a modular tree, for example) in three dimensions is an interesting question in its own right.

Within the IBM and compatibles world, adaptation to the forthcoming OS/2 operating system is an obvious migration path from the MSDOS environment. The arrival of IBM's promised "Presentation Manager" software should enhance the graphics operating environment. An alternative approach to enhance the current MSDOS environment is the use of an add-on "windowing" environment. In this regard, DESQview

---

10 These comments pertain only to the implementation of a visual interface. They do not necessarily pertain to the overall implementation of a model management system.

from Quarterdeck Office Systems is a very strong contender. Not only does it offer windowing capability, but it also provides extensive memory management and multi-tasking capabilities, both of which could be can be effectively utilized in a model management interface.

Rather than windowing, a two monitor system could also be devised. One monitor would be used to display and manipulate the graphics information. A second monitor could display textual data and allow direct access to the ORACLE RDBMS through the SQL*Forms utility. This is an ORACLE function that lets you design custom input "forms" for any application. These forms provide fast and easy data entry, updates, deletions, and queries to an ORACLE database. This would provide the best of both worlds, so to speak, allowing a truly integrated graphics and text environment.

However, perhaps the most attractive environment from a hardware perspective is the Apple Macintosh II. Based on a proven visual interface, the "MAC II" offers integrated windowing, menus, mouse-control, graphics primitives, and expansion memory. It provides adequate high resolution graphics and color without the confusion engendered by the multitude of add-on graphics devices available for the IBM PC's and compatibles. From a software viewpoint, its advantages are less apparent, although developing a model management interface under Apple's new HyperCard software certainly offers possibilities worth pursuing. The main drawback is

the availability of a suitable relational database system--in particular, one that supports SQL (Structured Query Language).

APPENDIX A

HALO Supported Graphics Devices

1. Amdex MAI
2. AT&T Display Enhancement Board
3. AT&T Image Capture Board
4. AT&T Indigenous Graphics Board
5. AT&T TARGA 8 and M8
6. AT&T TARGA 16
7. AT&T Video Display Adapter
8. Conographics Model 25
9. Conographics Model 40
10. Datacube IVG-128
11. Generic IBM Display Card
12. Hercules Monochrome Graphics
13. IBM 3270 PC with All Points Addressable Graphics Adapter
14. IBM Color Graphics Adapter
15. IBM Enhanced Graphics Adapter(EGA)
16. IBM PCjr
17. Imaging Technology FG-100-AT
18. Imaging Technology PC-Vision
19. Imagraph AGC 4
20. Imagraph AGC 8 (IM512P and IM1024P)
21. Metheus Omega/PC Display Processor
22. Micro Display Systems Genius VHR
23. New Media Graphics PC Overlay
24. Number Nine Revolution
25. Number Nine Revolution 2048x4
26. Quadram Palette Master
27. Quadram Quadscreen
28. Quadram Quadcolor II
29. Scion PC 640
30. Sigma Designs Color 400
31. STB Graphix Plus II
32. Tecmar Graphics Master

33. Texas Instruments Professional
34. Tseng Labs EVA & EVA/480
35. Virtual Raster Interface[11]
36. WYSE WY-700

---

11 This device driver is unique in that is does not control a specific board. Rather, it creates the model of a display device in the IBM PC user memory space. Such a virtual display device permits the user to create a display of any arbitrary resolution and then output the display to a hard copy device. It is thus possible to produce printouts of much higher resolution than the display device actually installed in the computer.

APPENDIX B

Structure Charts

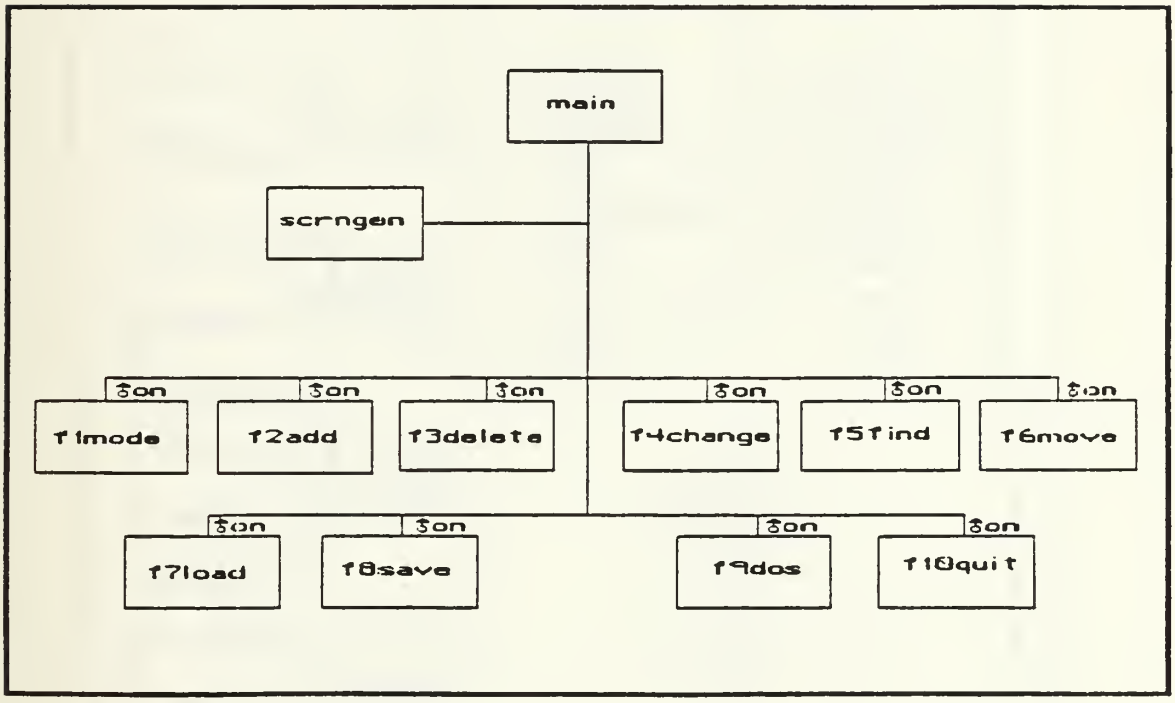

Figure 10: INTUITION (V1.1)

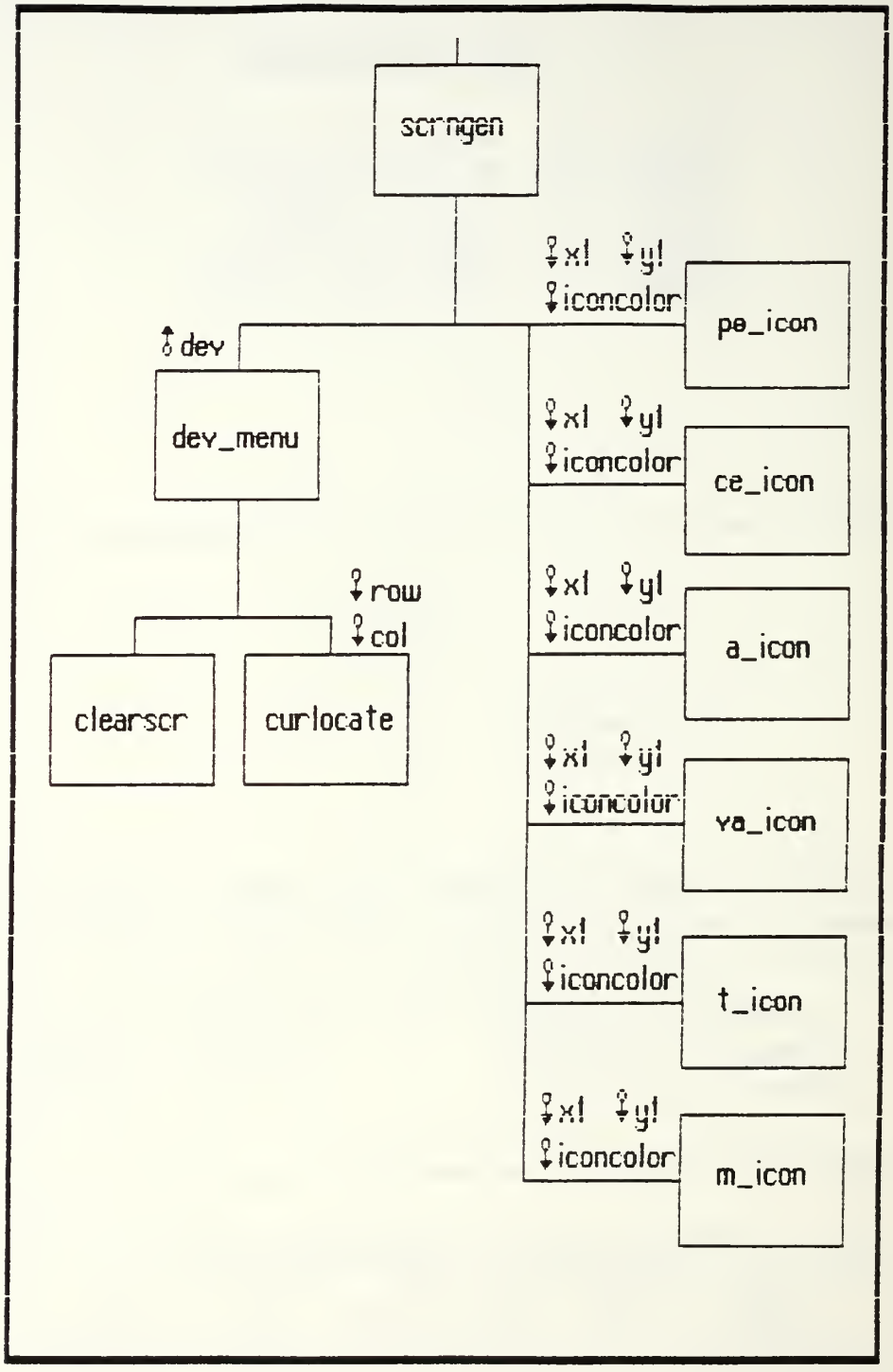

Figure 11: scrngen() module

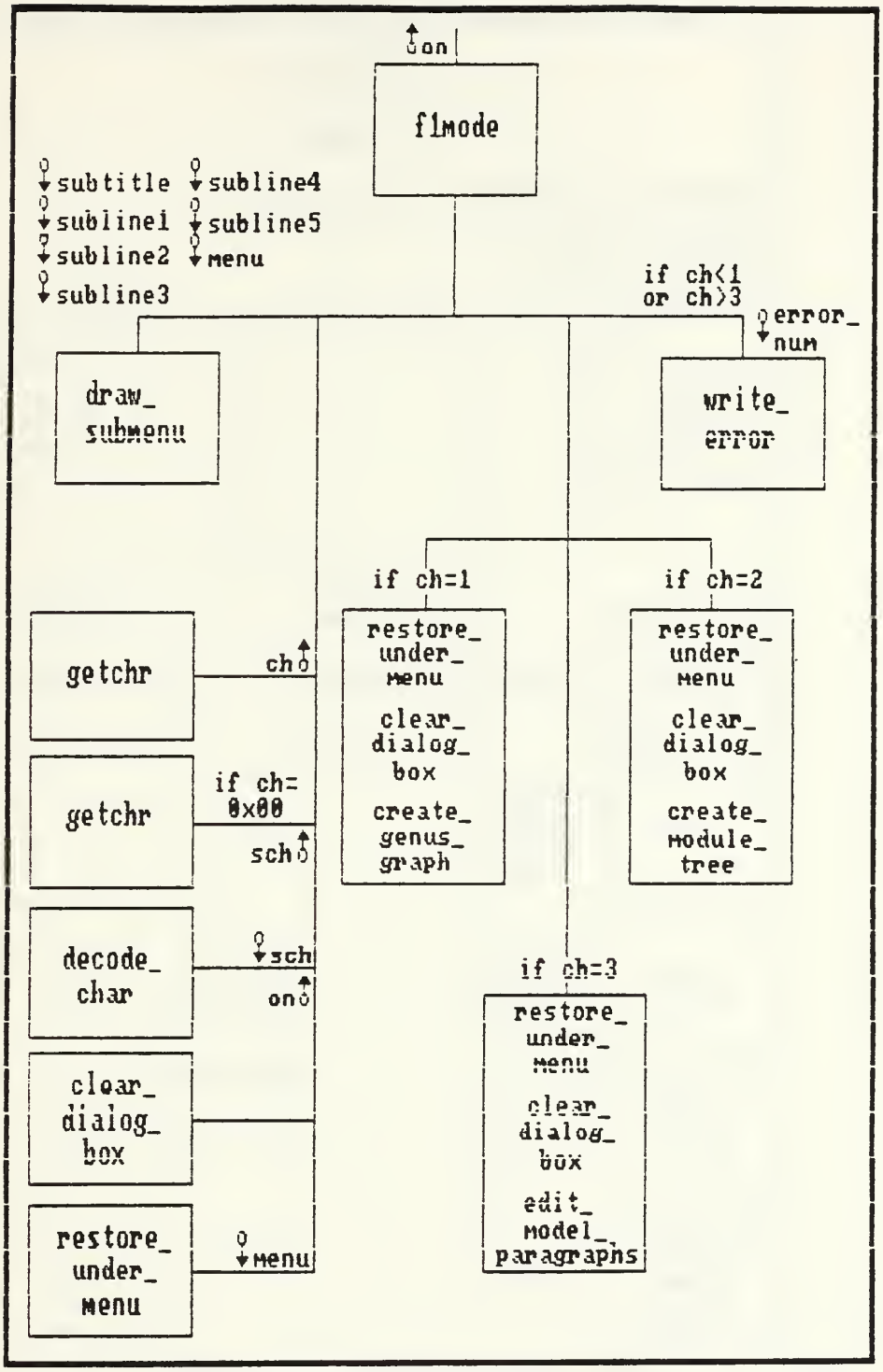

Figure 12: flmode() module

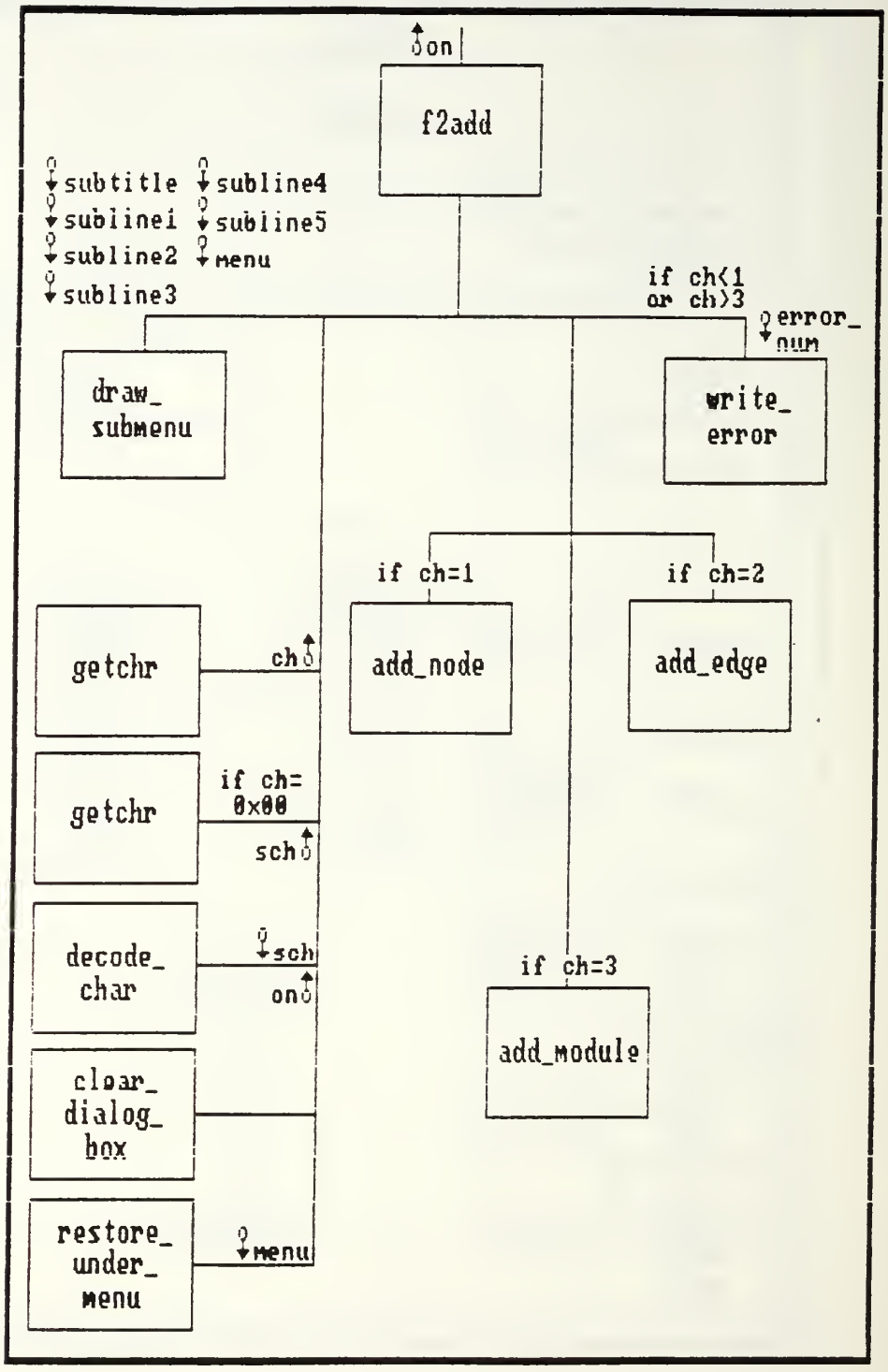

Figure 13: f2add() module

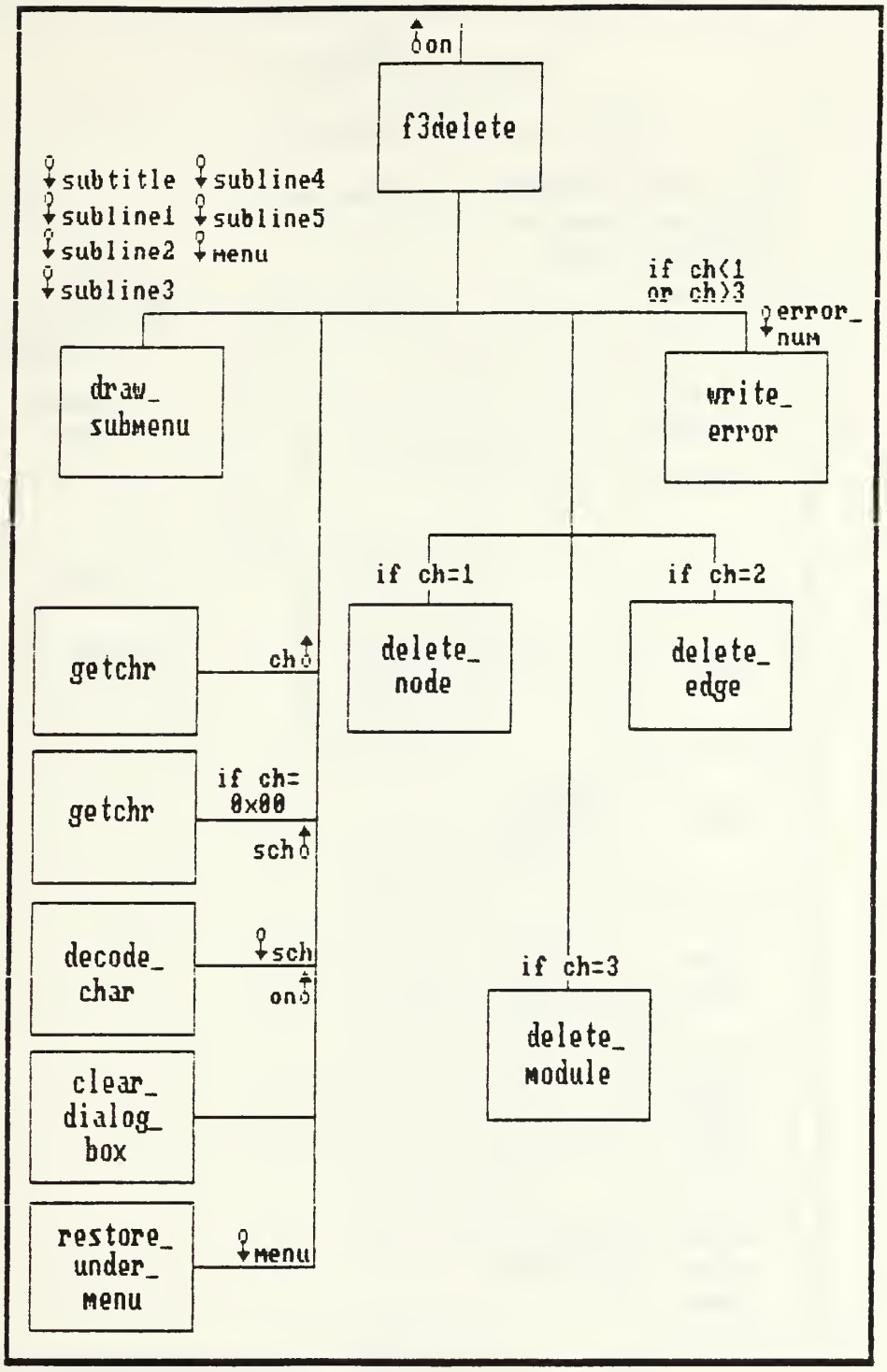

Figure 14: f3delete() module

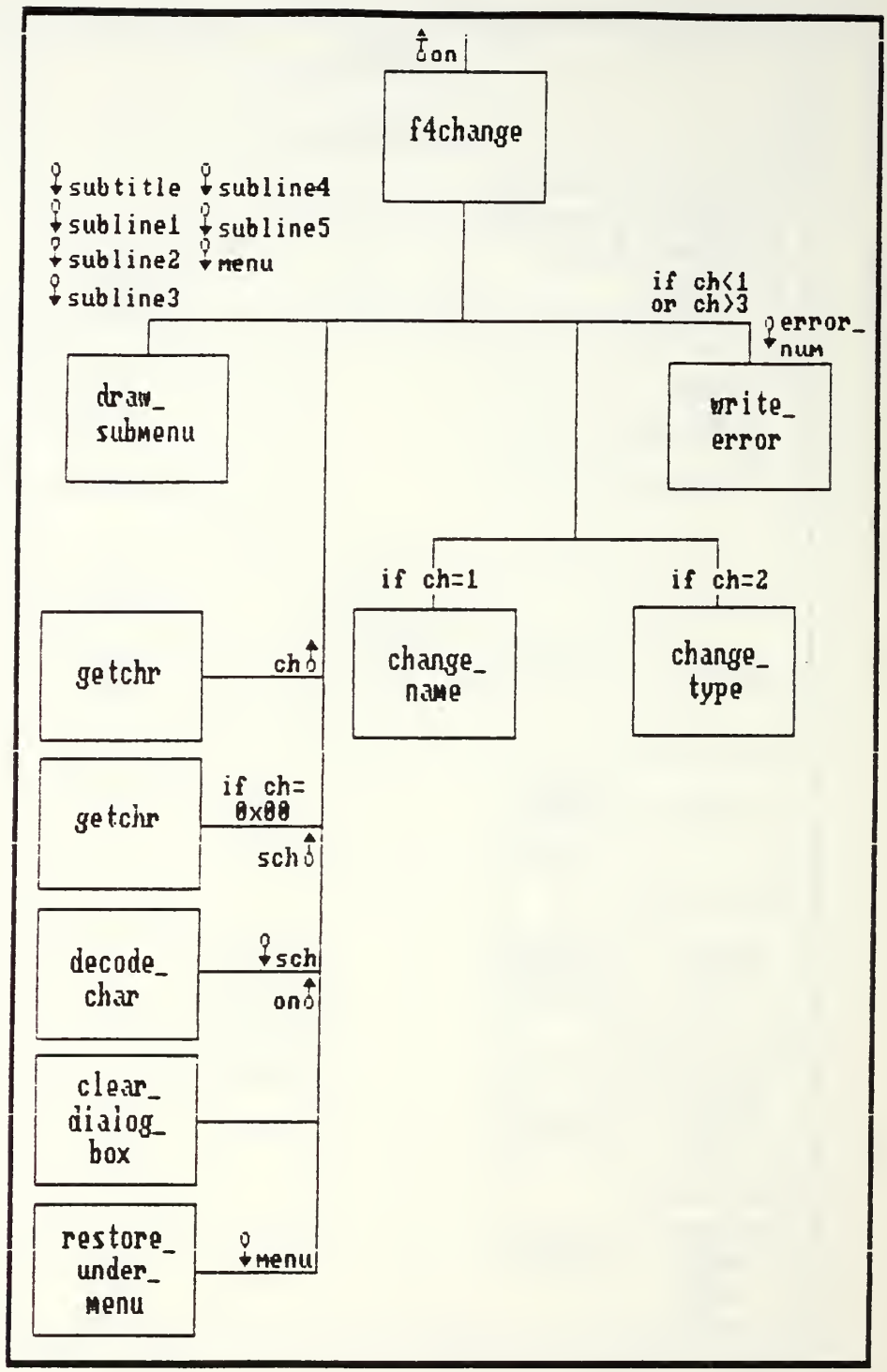

Figure 15: f4change() module

APPENDIX C

Program Listing

	File Name	Function Name
1.	intuit.h struct.h exstruct.h	a. none (header files)
2.	int.c	a. main()
3.	scrn.c	a. scrngen()
4.	menu.c	a. devmenu() b. clearscn() c. curlocate()
5.	icons.c	a. pe_icon() b. ce_icon() c. a_icon() d. va_icon() e. f_icon() f. t_icon() g. m_icon() h. arrow()
6.	util.c	a. bigport() b. littleport() c. getchr() d. decode_char() e. inside() f. draw_submenu() g. restore_under_menu() h. define_dot_test() i. write_error(); j. clear_dialog_box() k. clear_status_box() l. write_dialog() m. get_string() n. get_name() o. get_model_name() p. init_workspace()
7.	fun.c	a. f1mode() b. f2add() c. f3delete() d. f4change() e. f5find()

		f. f6move()
		g. f7load()
		h. f8save()
		i. f9dbms()
		j. f10quit()
8.	sfun1.c	a. create_genus_graph()
		b. add_node()
		c. add_edge()
		d. change_name()
		e. change_type()
9.	sfun2.c	a. create_module_tree()
		b. edit_model_paragraphs()
		c. add_module()
		d. delete_node()
		e. delete_edge()
		f. delete_module()

```
/*************************************************************
*  Name:      intuit.h                                       *
*  Purpose:   contains #defines for Intuition                *
*  Author:    David D. O'Dell                                *
*  Date:      15 December 1987                               *
*************************************************************/


#define MAXNODES        56    /*maximum nodes in workspace*/
#define BEGIN            0    /*starting point of edge*/
#define END              4    /*ending point of edge*/
#define MAXEDGES         5    /*maximun edges in and out of node*/

#define MAXPOINTS        6    /*maximum coord pairs for an edge*/

#define TRUE    1             /*boolean true is non-zero*/
#define FALSE   0             /*boolean false is zero*/

#define BLACK   0             /*standard colors*/
#define BLUE    1
#define GREEN   2
#define CYAN    3
#define RED     4
#define MAGENTA 5
#define BROWN   6
#define WHITE   7
#define GREY    8
#define BRIGHT_BLUE     9
#define BRIGHT_GREEN   10
#define BRIGHT_CYAN    11
#define BRIGHT_RED     12
#define BRIGHT_MAGENTA 13
#define YELLOW         14
#define BRIGHT_WHITE   15

#define SOLID          1       /*hatch style indices*/
#define DITHERED       2

#define WXMAX  8.05       /*set maximum world x-coordinate*/
#define WYMAX  5.05       /*set maximum world y-coordinate*/
#define WXMIN  0.0        /*set minimum world x-coordinate*/
#define WYMIN  0.0        /*set minimum world y-coordinate*/

#define BUTTONS  1             /*command button area*/
#define ICONS    2             /*icon area*/
#define WORKAREA 3             /*work area*/

/*following coordinates are given in world coord system*/

#define XLLSTATUS  0.0    /*status box lower left x-coord*/
#define YLLSTATUS  4.7    /*status box lower left y-coord*/
```

```
#define XURSTATUS  6.0   /*status box upper right x-coord*/
#define YURSTATUS  5.0   /*status box upper right y-coord*/

#define XLLROW     6.0   /*row box lower left x-coord*/
#define YLLROW     4.6   /*row box lower left y-coord*/
#define XURROW     7.0   /*row box upper right x-coord*/
#define YURROW     5.0   /*row box upper right y-coord*/

#define XLLCOL     7.0   /*col box lower left x-coord*/
#define YLLCOL     4.6   /*col box lower left y-coord*/
#define XURCOL     8.0   /*col box upper right x-coord*/
#define YURCOL     5.0   /*col box upper right y-coord*/

#define XLLICONS   6.0   /*icon area lower left x-coord*/
#define YLLICONS   1.8   /*icon area lower left y-coord*/
#define XURICONS   8.0   /*icon area upper right x-coord*/
#define YURICONS   4.6   /*icon area upper right y-coord*/

#define XLLCMDHDR  6.0   /*cmd hdr lower left x-coord*/
#define YLLCMDHDR  1.5   /*cmd hdr lower left y-coord*/
#define XURCMDHDR  8.0   /*cmd hdr upper right x-coord*/
#define YURCMDHDR  1.8   /*cmd hdr upper right y-coord*/

#define XLLBUTTONS 6.0   /*buttons lower left x-coord*/
#define YLLBUTTONS 0.0   /*buttons lower left y-coord*/
#define XURBUTTONS 8.0   /*buttons upper right x-coord*/
#define YURBUTTONS 1.5   /*buttons upper right y-coord*/
#define XLLDIALOG  0.0   /*dialog box lower left x-coord*/
#define YLLDIALOG  0.0   /*dialog box lower left y-coord*/
#define XURDIALOG  6.0   /*dialog box upper right x-coord*/
#define YURDIALOG  0.6   /*dialog box upper right y-coord*/

#define XLLWORK    0.0   /*workspace lower left x-coord*/
#define YLLWORK    0.6   /*workspace lower left y-coord*/
#define XURWORK    6.0   /*workspace upper right x-coord*/
#define YURWORK    4.7   /*workspace upper right y-coord*/

#define XLLPE      6.0   /*PE lower left x-coord*/
#define YLLPE      3.9   /*PE lower left y-coord*/
#define XURPE      7.0   /*PE upper right x-coord*/
#define YURPE      4.6   /*PE upper right y-coord*/
#define XLLCE      7.0   /*CE lower left x-coord*/
#define YLLCE      3.9   /*CE lower left y-coord*/
#define XURCE      8.0   /*CE upper right x-coord*/
#define YURCE      4.6   /*CE upper right y-coord*/

#define XLLA       6.0   /*A lower left x-coord*/
#define YLLA       3.2   /*A lower left y-coord*/
#define XURA       7.0   /*A upper right x-coord*/
#define YURA       3.9   /*A upper right y-coord*/

#define XLLVA      7.0   /*VA lower left x-coord*/
```

```
#define YLLVA       3.2   /*VA lower left y-coord*/
#define XURVA       8.0   /*VA upper right x-coord*/
#define YURVA       3.9   /*VA upper right y-coord*/

#define XLLF        6.0   /*F lower left x-coord*/
#define YLLF        2.5   /*F lower left y-coord*/
#define XURF        7.0   /*F upper right x-coord*/
#define YURF        3.2   /*F upper right y-coord*/

#define XLLT        7.0   /*T lower left x-coord*/
#define YLLT        2.5   /*T lower left y-coord*/
#define XURT        8.0   /*T upper right x-coord*/
#define YURT        3.2   /*T upper right y-coord*/

#define XLLM        6.0   /*M lower left x-coord*/
#define YLLM        1.8   /*M lower left y-coord*/
#define XURM        8.0   /*M upper right x-coord*/
#define YURM        2.5   /*M upper right y-coord*/

#define XLLF1       6.0   /*F1 lower left x-coord*/
#define YLLF1       1.2   /*F1 lower left y-coord*/
#define XURF1       7.0   /*F1 upper right x-coord*/
#define YURF1       1.5   /*F1 upper right y-coord*/

#define XLLF2       6.0   /*F2 lower left x-coord*/
#define YLLF2       0.9   /*F2 lower left y-coord*/
#define XURF2       7.0   /*F2 upper right x-coord*/
#define YURF2       1.2   /*F2 upper right y-coord*/

#define XLLF3       6.0   /*F3 lower left x-coord*/
#define YLLF3       0.6   /*F3 lower left y-coord*/
#define XURF3       7.0   /*F3 upper right x-coord*/
#define YURF3       0.9   /*F3 upper right y-coord*/

#define XLLF4       6.0   /*F4 lower left x-coord*/
#define YLLF4       0.3   /*F4 lower left y-coord*/
#define XURF4       7.0   /*F4 upper right x-coord*/
#define YURF4       0.6   /*F4 upper right y-coord*/

#define XLLF5       6.0   /*F5 lower left x-coord*/
#define YLLF5       0.0   /*F5 lower left y-coord*/
#define XURF5       7.0   /*F5 upper right x-coord*/
#define YURF5       0.3   /*F5 upper right y-coord*/

#define XLLF6       7.0   /*F6 lower left x-coord*/
#define YLLF6       1.2   /*F6 lower left y-coord*/
#define XURF6       8.0   /*F6 upper right x-coord*/
#define YURF6       1.5   /*F6 upper right y-coord*/

#define XLLF7       7.0   /*F7 lower left x-coord*/
#define YLLF7       0.9   /*F7 lower left y-coord*/
#define XURF7       8.0   /*F7 upper right x-coord*/
```

```
#define YURF7       1.2    /*F7 upper right y-coord*/

#define XLLF8       7.0    /*F8 lower left x-coord*/
#define YLLF8       0.6    /*F8 lower left y-coord*/
#define XURF8       8.0    /*F8 upper right x-coord*/
#define YURF8       0.9    /*F8 upper right y-coord*/

#define XLLF9       7.0    /*F9 lower left x-coord*/
#define YLLF9       0.3    /*F9 lower left y-coord*/
#define XURF9       8.0    /*F9 upper right x-coord*/
#define YURF9       0.6    /*F9 upper right y-coord*/

#define XLLF10      7.0    /*F10 lower left x-coord*/
#define YLLF10      0.0    /*F10 lower left y-coord*/
#define XURF10      8.0    /*F10 upper right x-coord*/
#define YURF10      0.3    /*F10 upper right y-coord*/

#define TEXTHT      1      /*dot text height x 8 pixles*/
#define TEXTWD      1      /*dot text width x 8 pixels*/
#define HORIZONTAL  0.0    /*horizontal text path*/
#define ROTATE_90   1.0    /*vertical text path*/
#define ROTATE_180  3.0    /*up-side-down text path*/
#define ROTATE_270  4.0    /*vertical text path*/
#define BORDER      1      /*border around dot text*/
#define NOBORDER    0      /*no border around dot text*/
```

```
/*************************************************************
*  Name:     struct.h                                        *
*  Purpose:  contains data structures for ORACLE tables and  *
*            to hold coordinates for model nodes and edges.  *
*  Author:   David D. O'Dell                                 *
*  Date:     10 March 1988                                   *
*************************************************************/


        struct table
        {
             char *ename;              /*element name*/
             char *etype;              /*element type*/
             char *dname;              /*descriptive name*/
             char *date_added;         /*date node created*/
             char *last_mod;           /*date last modified*/
             int nmods;                /*number of mods*/
             char *idx;                /*index*/
             char *idx_stmt;           /*index statement*/
             char *grange;             /*generic range*/
             char *grule;              /*generic rule*/
             char *comments;           /*interpretation*/

             char *rtype;              /*relationship type*/
             char *elname[5];          /*calling element*/
             char *eltype[5];          /*calling type*/
             char *e2name[5];          /*called element*/
             char *e2type[5];          /*called type*/
             char *acc_meth;           /*access method and*/
             int freq;                 /*frequency always null*/
             int rel_pos;        /*position in monotone order*/
        };

        struct node
        {
             int used;         /*TRUE is used-FALSE is unused*/
             char *ename;      /*link to ORACLE tables*/
             char *etype;      /*node type*/
             float nodex;      /*x-coord of cell center*/
             float nodey;      /*y-coord of cell center*/
             int out;          /*number of edges out used*/
             int in;           /*number of edges in used*/
             float edgex[5][5];      /*x-coord of edges out 1-5*/
             float edgey[5][5];      /*y-coord of edges out 1-5*/
             int cnode[5];           /*number of called by node*/
             int cedge[5];           /*number of called by edge*/
        };

        /*element is a table structure*/
        struct table element[MAXNODES];
```

```
/*cell is a node structure*/
struct node cell[MAXNODES];

int used_nodes;          /*number of used nodes*/
int active_node;         /*currently selected node*/
char *model_name;
```

```
/*************************************************************
*  Name:     exstruct.h                                      *
*  Purpose:  contains external data declarations corresponding *
*            to those in struct.h.                           *
*  Author:   David D. O'Dell                                 *
*  Date:     10 March 1988                                   *
*************************************************************/


        extern struct table
        {
             char *ename;              /*element name*/
             char *etype;              /*element type*/
             char *dname;              /*descriptive name*/
             char *date_added;         /*date node created*/
             char *last_mod;           /*date last modified*/
             int nmods;                /*number of modifications*/
             char *idx;                /*index*/
             char *idx_stmt;           /*index statement*/
             char *grange;             /*generic range*/
             char *grule;              /*generic rule*/
             char *comments;           /*interpretation*/

             char *rtype;              /*relationship type*/
             char *elname[5];          /*calling element*/
             char *eltype[5];          /*calling type*/
             char *e2name[5];          /*called element*/
             char *e2type[5];          /*called type*/
             char *acc_meth;           /*access method always
                                         null*/
             int freq;                 /*frequency always null*/
             int rel_pos;              /*position in monotone
                                         order*/
        };

        extern struct node
        {
             int used;                /*TRUE is used-FALSE is
                                        unused*/
             char *ename;             /*link to ORACLE tables*/
             char *etype;             /*node type*/
             float nodex;             /*x-coord of cell center*/
             float nodey;             /*y-coord of cell center*/
             int out;                 /*number of edges out used*/
             int in;                  /*number of edges in used*/
             float edgex[5][5];       /*x-coord of edges out 1-5*/
             float edgey[5][5];       /*y-coord of edges out 1-5*/
             int cnode[5];            /*number of called by node*/
             int cedge[5];            /*number of called by edge*/
        };
```

```
/*element is a table structure*/
extern struct table element[MAXNODES];

/*cell is a node structure*/
extern struct node cell[MAXNODES];

extern int used_nodes;          /*number of used nodes*/
extern int active_node;         /*currently selected node*/
extern char *model_name;
```

```
/*****************************************************************
*  Name:     intuition.c                                          *
*  Purpose:  to allow the user to enter a structured model        *
*  Author:   David D. O'Dell                                      *
*  Date:     29 December 1987                                     *
*****************************************************************/


#include "stdio.h"
#include "intuit.h"
#include "struct.h"


main()
{
     char getchr();  /*function to input keyboard character*/

     float x1, y1, x2, y2;    /*graphic coordinate variables*/

     float cx, cy;            /*x-hair cursor coordinates*/

     float button_height;     /*command button height*/

     float button_width;      /*command button width*/

     float hheight, hwidth;   /*height and width of x-hair
                                cursor*/

     int sw;                  /*exclusive-or mode switch*/

     int status;              /*termination status variable*/

     int color;               /*color and hatchstyle*/

     int on;                  /*determines command button
 selected*/

 /*-------------------------------------------------------------*/

     status=0;            /*normal termination status*/

     scrngen();           /*draw display screen*/

     bigport();           /*set full screen viewport*/

     color=WHITE;
     setcolor(&color);

     button_height=YURF1-YLLF1;    /*calc command button height*/

     button_width=XURF1-XLLF1;     /*calc command button width*/
```

```
     hheight=button_height/2.0;      /*height of x-hair cursor*/
     hwidth=button_width/2.0;        /*width of x-hair cursor*/
     color=WHITE;                    /*color of x-hair cursor*/
     inithcur(&hheight, &hwidth, &color); /*init x-hair cursor*/

     cx=XLLBUTTONS+hwidth;          /*put x-hair cursor in the*/
     cy=YURBUTTONS-hheight;         /*upper right corner of the*/
     movhcurabs(&cx, &cy);          /*command button area*/
     delhcur();                     /*don't show x-hair cursor*/


     on=1;                            /*initialize to f1button*/
     x1=0.0; y1=0.0; x2=0.0; y2=0.0;  /*initialize bar coord*/

while(TRUE)
{
     sw=1;              /*1=on*/
     setxor(&sw);       /*turn on exclusive-or mode*/

     bar(&x1,&y1,&x2,&y2);  /*turn old command button off*/

     switch(on)                       /*changes button color*/
     {
          case 1:   x1=XLLF1; y1=YLLF1; /*low lt coord of
                                          F1button*/
                    x2=XURF1; y2=YURF1; /*up rt coords of
                                          F1button*/
                    break;
          case 2:   x1=XLLF2; y1=YLLF2; /*low lt coord of
                                          F2button*/
                    x2=XURF2; y2=YURF2; /*up rt coords of
                                          F2button*/
                    break;
          case 3:   x1=XLLF3; y1=YLLF3; /*low lt coord of
                                          F3button*/
                    x2=XURF3; y2=YURF3; /*up rt coords of
                                          F3button*/
                    break;
          case 4:   x1=XLLF4; y1=YLLF4; /*low lt coord of
                                          F4button*/
                    x2=XURF4; y2=YURF4; /*up rt coords of
                                          F4button*/
                    break;
          case 5:   x1=XLLF5; y1=YLLF5; /*low lt coord of
                                          F5button*/
                    x2=XURF5; y2=YURF5; /*up rt coords of
                                          F5button*/
                    break;

          case 6:   x1=XLLF6; y1=YLLF6; /*low lt coord of
                                          F6button*/
```

```
                x2=XURF6; y2=YURF6; /*up rt coords of
                                       F6button*/
                break;
     case 7:    x1=XLLF7; y1=YLLF7; /*low lt coord of
                                       F7button*/
                x2=XURF7; y2=YURF7; /*up rt coords of
                                       F7button*/
                break;
     case 8:    x1=XLLF8; y1=YLLF8; /*low lt coord of
                                       F8button*/
                x2=XURF8; y2=YURF8; /*up rt coords of
                                       F8button*/
                break;
     case 9:    x1=XLLF9; y1=YLLF9; /*low lt coord of
                                       F9button*/
                x2=XURF9; y2=YURF9; /*up rt coords of
                                       F9button*/
                break;
     case 10:   x1=XLLF10; y1=YLLF10; /*low lt coord of
                                         F10button*/
                x2=XURF10; y2=YURF10; /*up rt coords of
                                         F10button*/
                break;
     default:   break;
}  /*end switch*/

bar(&x1,&y1,&x2,&y2);  /*turn new command button on*/

sw=0;                  /*0=off*/
setxor(&sw);           /*turn off exclusive-or mode*/

switch(on)             /*selects function*/
{
     case 1:    on=f1mode();
                break;
     case 2:    on=f2add();
                break;
     case 3:    on=f3delete();
                break;
     case 4:    on=f4change();
                break;
     case 5:    on=f5find();
                break;
     case 6:    on=f6move();
                break;
     case 7:    on=f7load();
                break;
     case 8:    on=f8save();
                break;
     case 9:    on=f9dbms();
                break;
```

```
                case 10:  on=f10quit();
                          break;
                default:  closegraphics();
                          exit(status);
                          break;
            }  /*end switch*/

        }  /*while true*/

        } /*end main*/
```

```
/*****************************************************************
*  Name:     scrn.c                                              *
*  Purpose:  to create the background screen for Intuition       *
*  Author:   David D. O'Dell                                     *
*  Date:     15 December 1987                                    *
*****************************************************************/


#include "intuit.h"


scrngen()
{
     static char device1[] = "HALOIBM.DEV"; /*CGA raster dev*/

     static char device2[] = "HALOIBMG.DEV"; /*generic dev*/

     static char device3[] = "HALOIBME.DEV"; /*EGA raster dev*/

     static char device4[] = "HALOSIGM.DEV"; /*Sigma 400 dev*/

     static char title1[] = "TYPE:";         /*screen titles*/

     static char title2[] = "NAME:";

     static char title3[] = "ROW:";

     static char title4[] = "COL:";

     static char title5[] = "COMMAND  BUTTONS";

     static char button1[] = "F1 Mode ";      /*command button
                                                labels*/

     static char button2[] = "F2 Add  ";

     static char button3[] = "F3 Del  ";

     static char button4[] = "F4 Chg  ";

     static char button5[] = "F5 Find ";

     static char button6[] = "F6 Move ";

     static char button7[] = "F7 Load ";

     static char button8[] = "F8 Save ";

     static char button9[] = "F9 DBMS ";

     static char button10[] = "F10 Quit";
```

```
static char icon1[] = "PE";       /*icon labels-primitive
                                    entity*/

static char icon2[] = "CE";       /*compound entity*/

static char icon3[] = "A";        /*attribute element*/

static char icon4[] = "VA";       /*variable attribute
                                    element*/

static char icon5[] = "F";        /*function element*/

static char icon6[] = "T";        /*test element*/

static char icon7[] = "M_";        /*module*/


float x1, y1, x2, y2;    /*coordinate variables*/

float tx, ty;            /*text coordinate variables*/

int status;              /*zero-normal program termination*/


int row, col;            /*location of cursor on text
                           screen*/

int i;                   /*loop variable*/

int color, style;        /*color and hatchstyle*/

int mode;                /*graphics mode*/

int foreground;          /*foreground color*/

int border;              /*border color*/

int iconcolor;           /*display color of icon*/

int palette;             /*palette number*/

int index;               /*color index for EGA dev*/

int dev;                 /*type of graphics card selected*/

int height;              /*dot text height*/

int width;               /*dot text width*/

int path;                /*dot text display direction*/

int textmode;            /*dot text display mode*/
```

```
     int textforeground;       /*dot text foreground color*/

     int textbackground;       /*dot text background color*/

/*-------------------------------------------------------------*/

     status=0;              /*normal program termination status*/

     dev=devmenu();     /*display menu and get graphics device*/

     if(dev==1)         /*set graphics device and mode*/
     {
          setdev(device1);     /*IBM CGA device*/
          mode=1;              /*640 x 200 - 2 colors*/
          foreground=WHITE;    /*set foreground color*/
          palette=0;      /*set unused dummy value in palette*/
          setipal(&foreground, &palette);  /*set IBM palette*/
     }
     else if(dev==2)
     {
          setdev(device2);     /*IBM generic CGA device*/
          mode=1;              /*640 x 200 - 2 colors*/
          foreground=WHITE;    /*set foreground color*/
          palette=0;     /*set unused dummy value in palette*/
          setipal(&foreground,&palette);  /*set IBM palette*/
     }
     else if(dev==3)
     {
          setdev(device3);     /*IBM EGA device*/
          mode=4;              /*640 x 385 - 16 colors*/
          for(i=0; i<=15; i++)
          {
               index=i;        /*for this EGA color index...*/
               color=i;        /*this is the color.*/
               setxpal(&index, &color);   /*set the EGA palette*/

          } /*end for*/
     }
     else if(dev==4)
     {
          setdev(device4);     /*Sigma Designs 400 device*/
          mode=3;              /*640 x 400 - 16 colors*/
          border=BLACK;        /*set border color*/
          palette=0;           /*set unused dummy palette value*/
          setipal(&border, &palette);
     }
     else if(dev==5)
     {
          clearscn();          /*clear the text screen*/
```

```
     row=0; col=0;        /*upper left corner of screen*/
     curlocate(row,col); /*move cursor*/
     exit(status);        /*terminate program*/
}  /*end if-else*/

initgraphics(&mode);      /*initialize graphics mode and
                            clear the graphics screen*/

x1=WXMIN; y1=WYMIN; x2=WXMAX; y2=WYMAX;
setworld(&x1, &y1, &x2, &y2);     /*set world coord system*/


if((dev==1) || (dev==2))          /*if two color mode*/
     height=TEXTHT;
else
     height=(2*TEXTHT);           /*if 16 color mode*/
width=TEXTWD;                     /*text width in pixels*/
path=HORIZONTAL;                  /*text display direction*/

textmode=NOBORDER;                /*text display mode*/
settext(&height, &width, &path, &textmode);   /*set
                                                attributes*/
color=WHITE;
setcolor(&color);   /*set active drawing color*/

x1=XLLSTATUS; y1=YLLSTATUS;   /*low lt coord of status box*/

x2=XURSTATUS; y2=YURSTATUS;   /*up rt coords of status box*/

box(&x1, &y1, &x2, &y2);      /*draw status box*/

x1=XLLROW; y1=YLLROW;         /*low lt coord of row box*/
x2=XURROW; y2=YURROW;         /*up rt coords of row box*/
box(&x1, &y1, &x2, &y2);      /*draw row box*/

x1=XLLCOL; y1=YLLCOL;         /*low lt coord of col box*/
x2=XURCOL; y2=YURCOL;         /*up rt coords of col box*/
box(&x1, &y1, &x2, &y2);      /*draw col box*/

x1=XLLCMDHDR; y1=YLLCMDHDR;   /*low lt coord of cmd hdr*/
x2=XURCMDHDR; y2=YURCMDHDR;   /*up rt coords of cmd hdr*/
box(&x1, &y1, &x2, &y2);      /*draw commands header*/

x1=XLLDIALOG; y1=YLLDIALOG;   /*low lt coord of dialog box*/

x2=XURDIALOG; y2=YURDIALOG;   /*up rt coords of dialog box*/

box(&x1, &y1, &x2, &y2);      /*draw dialog box*/
```

```
if((dev==1) || (dev==2))        /*if two color mode*/
{
     style=DITHERED;            /*creates GREY color*/
     sethatchstyle(&style);     /*set active hatch style*/
}
else                            /*if 16 color mode*/
{
     color=GREY;
     setcolor(&color);          /*set active drawing color*/
}  /*end if-else*/

x1=XLLICONS; y1=YLLICONS;       /*low lt coord of icon area*/
x2=XURICONS; y2=YURICONS;       /*up rt coords of icon area*/
bar(&x1, &y1, &x2, &y2);        /*draw icon area*/

if((dev==3) || (dev==4))        /*if 16 color mode*/
{
     color=BROWN;
     setcolor(&color);          /*set active drawing color*/
}
else                            /*if two color mode*/
{
     style=SOLID;
     sethatchstyle(&style);   /*set active hatch style*/
     color=BLACK;
     setcolor(&color);       /*set active drawing color*/
}  /*end if-else*/

x1=XLLBUTTONS; y1=YLLBUTTONS; /*low lt coords  button area*/

x2=XURBUTTONS; y2=YURBUTTONS; /*up rt coord of button area*/

bar(&x1, &y1, &x2, &y2);        /*draw button area*/

color=WHITE;
setcolor(&color);               /*set active drawing color*/

x1=XLLWORK; y1=YLLWORK;         /*low lt coord of workspace*/
x2=XURWORK; y2=YURWORK;         /*up rt coords of workspace*/
bar(&x1, &y1, &x2, &y2);        /*draw workspace*/

x1=XLLPE; y1=YLLPE;             /*low lt coord of PE area*/
x2=XURPE; y2=YURPE;             /*up rt coords of PE area*/
box(&x1, &y1, &x2, &y2);        /*draw PE frame*/

x1=XLLCE; y1=YLLCE;             /*low lt coord of CE area*/
x2=XURCE; y2=YURCE;             /*up rt coords of CE area*/
box(&x1, &y1, &x2, &y2);        /*draw CE frame*/

x1=XLLA; y1=YLLA;               /*low lt coord of A area*/
x2=XURA; y2=YURA;               /*up rt coords of A area*/
box(&x1, &y1, &x2, &y2);        /*draw A frame*/
```

```
x1=XLLVA; y1=YLLVA;          /*low lt coord of VA area*/
x2=XURVA; y2=YURVA;          /*up rt coords of VA area*/
box(&x1, &y1, &x2, &y2);     /*draw VA frame*/

x1=XLLF; y1=YLLF;            /*low lt coord of F area*/
x2=XURF; y2=YURF;            /*up rt coords of F area*/
box(&x1, &y1, &x2, &y2);     /*draw F frame*/

x1=XLLT; y1=YLLT;            /*low lt coord of T area*/
x2=XURT; y2=YURT;            /*up rt coords of T area*/
box(&x1, &y1, &x2, &y2);     /*draw T frame*/

x1=XLLM; y1=YLLM;            /*low lt coord of M area*/
x2=XURM; y2=YURM;            /*up rt coords of M area*/
box(&x1, &y1, &x2, &y2);     /*draw M frame*/

x1=XLLF1; y1=YLLF1;          /*low lt coord of F1 button*/
x2=XURF1; y2=YURF1;          /*up rt coords of F1 button*/
box(&x1, &y1, &x2, &y2);     /*draw F1 frame*/

x1=XLLF2; y1=YLLF2;          /*low lt coord of F2 button*/
x2=XURF2; y2=YURF2;          /*up rt coords of F2 button*/
box(&x1, &y1, &x2, &y2);     /*draw F2 frame*/

x1=XLLF3; y1=YLLF3;          /*low lt coord of F3 button*/
x2=XURF3; y2=YURF3;          /*up rt coords of F3 button*/
box(&x1, &y1, &x2, &y2);     /*draw F3 frame*/

x1=XLLF4; y1=YLLF4;          /*low lt coord of F4 button*/
x2=XURF4; y2=YURF4;          /*up rt coords of F4 button*/
box(&x1, &y1, &x2, &y2);     /*draw F4 frame*/

x1=XLLF5; y1=YLLF5;          /*low lt coord of F5 button*/
x2=XURF5; y2=YURF5;          /*up rt coords of F5 button*/
box(&x1, &y1, &x2, &y2);     /*draw F5 frame*/

x1=XLLF6; y1=YLLF6;          /*low lt coord of F6 button*/
x2=XURF6; y2=YURF6;          /*up rt coords of F6 button*/
box(&x1, &y1, &x2, &y2);     /*draw F6 frame*/

x1=XLLF7; y1=YLLF7;          /*low lt coord of F7 button*/
x2=XURF7; y2=YURF7;          /*up rt coords of F7 button*/
box(&x1, &y1, &x2, &y2);     /*draw F7 frame*/

x1=XLLF8; y1=YLLF8;          /*low lt coord of F8 button*/
x2=XURF8; y2=YURF8;          /*up rt coords of F8 button*/
box(&x1, &y1, &x2, &y2);     /*draw F8 frame*/

x1=XLLF9; y1=YLLF9;          /*low lt coord of F9 button*/
x2=XURF9; y2=YURF9;          /*up rt coords of F9 button*/
box(&x1, &y1, &x2, &y2);     /*draw F9 frame*/
```

```
x1=XLLF10; y1=YLLF10;          /*low lt coord of F10 button*/

x2=XURF10; y2=YURF10;          /*up rt coords of F10 button*/

box(&x1, &y1, &x2, &y2);       /*draw F10 frame*/

textforeground=WHITE; textbackground=BLACK;  /*dot text
                                               colors*/
settextclr(&textforeground, &textbackground);  /*set
                                                 colors*/

tx=XLLSTATUS+0.2; ty=YLLSTATUS+0.05; /*text cursor coords*/
movtcurabs(&tx, &ty);                /*move text cursor*/
text(title1);                        /*label for type of
                                       model*/

tx=XLLSTATUS+4.0; ty=YLLSTATUS+0.05; /*text cursor coords*/
movtcurabs(&tx, &ty);                /*move text cursor*/
text(title2);                        /*label for name of
                                       model*/

tx=XLLROW+0.2; ty=YLLROW+0.1;        /*text cursor coords*/
movtcurabs(&tx, &ty);                /*move text cursor*/
text(title3);                        /*row label*/

tx=XLLCOL+0.2; ty=YLLCOL+0.1;        /*text cursor coords*/
movtcurabs(&tx, &ty);                /*move text cursor*/
text(title4);                        /*column label*/

tx=XLLCMDHDR+0.2; ty=YLLCMDHDR+0.05; /*text cursor coords*/
movtcurabs(&tx, &ty);                /*move text cursor*/
text(title5);                        /*command header
                                       label*/

if((dev==3) || (dev==4))       /*if 16 color mode*/
     textbackground=BROWN;
else                           /*if two color mode*/
     textbackground=BLACK;

tx=XLLF1+0.1; ty=YLLF1+0.05;         /*text cursor coords*/
movtcurabs(&tx, &ty);                /*move text cursor*/
text(button1);                       /*F1 label*/

tx=XLLF2+0.1; ty=YLLF2+0.05;         /*text cursor coords*/
movtcurabs(&tx, &ty);                /*move text cursor*/
text(button2);                       /*F2 label*/

tx=XLLF3+0.1; ty=YLLF3+0.05;         /*text cursor coords*/
movtcurabs(&tx, &ty);                /*move text cursor*/
text(button3);                       /*F3 label*/

tx=XLLF4+0.1; ty=YLLF4+0.05;         /*text cursor coords*/
```

```
movtcurabs(&tx, &ty);                   /*move text cursor*/
text(button4);                          /*F4 label*/

tx=XLLF5+0.1; ty=YLLF5+0.05;            /*text cursor coords*/
movtcurabs(&tx, &ty);                   /*move text cursor*/
text(button5);                          /*F5 label*/

tx=XLLF6+0.1; ty=YLLF6+0.05;            /*text cursor coords*/
movtcurabs(&tx, &ty);                   /*move text cursor*/
text(button6);                          /*F6 label*/

tx=XLLF7+0.1; ty=YLLF7+0.05;            /*text cursor coords*/
movtcurabs(&tx, &ty);                   /*move text cursor*/
text(button7);                          /*F7 label*/

tx=XLLF8+0.1; ty=YLLF8+0.05;            /*text cursor coords*/
movtcurabs(&tx, &ty);                   /*move text cursor*/
text(button8);                          /*F8 label*/

tx=XLLF9+0.1; ty=YLLF9+0.05;            /*text cursor coords*/
movtcurabs(&tx, &ty);                   /*move text cursor*/
text(button9);                          /*F9 label*/

tx=XLLF10+0.1; ty=YLLF10+0.05;          /*text cursor coords*/
movtcurabs(&tx, &ty);                   /*move text cursor*/
text(button10);                         /*F10 label*/

iconcolor=GREEN;                    /*primitive entity is
                                      green*/
x1=XLLPE; y1=YLLPE;                 /*lower left coords of icon
                                      area*/
pe_icon(x1, y1, iconcolor);         /*draw primitive entity*/

textforeground=BLACK; textbackground=iconcolor;  /*dot text
                                                   colors*/
settextclr(&textforeground, &textbackground);  /*set
                                                 colors*/
tx=x1+0.4; ty=y1+0.25;              /*primitive entity label
                                      coords*/
movtcurabs(&tx, &ty);               /*move text cursor*/
text(icon1);                        /*label primitive entity*/

iconcolor=BROWN;                    /*compound entity is
                                      brown*/
x1=XLLCE; y1=YLLCE;                 /*lower left coords of icon
                                      area*/
ce_icon(x1, y1, iconcolor);         /*draw compound entity*/

textforeground=BLACK; textbackground=iconcolor;  /*dot text
                                                   colors*/
settextclr(&textforeground, &textbackground);    /*set
                                                   colors*/
```

```
tx=x1+0.4; ty=y1+0.25;          /*compound entity label
                                  coords*/
movtcurabs(&tx, &ty);           /*move text cursor*/
text(icon2);                    /*label compound entity*/

iconcolor=CYAN;                 /*attribute elemet is
                                  cyan*/
x1=XLLA; y1=YLLA;               /*lower left coords of icon
                                  area*/
a_icon(x1, y1, iconcolor);      /*draw attribute element*/

textforeground=BLACK; textbackground=iconcolor;  /*dot text
                                                   colors*/
settextclr(&textforeground, &textbackground);    /*set
                                                   colors*/
tx=x1+0.45; ty=y1+0.25;         /*attribute element label
                                  coords*/
movtcurabs(&tx, &ty);           /*move text cursor*/
text(icon3);                    /*label attribute element*/


iconcolor=BLUE;                 /*variable attribute is
                                  blue*/
x1=XLLVA; y1=YLLVA;             /*lower left coords of icon
                                  area*/
va_icon(x1, y1, iconcolor);     /*draw variable attribute
                                  element*/

textforeground=BLACK; textbackground=iconcolor;  /*dot text
                                                   colors*/
settextclr(&textforeground, &textbackground);    /*set
                                                   colors*/
tx=x1+0.4; ty=y1+0.25;          /*variable attribute label
                                  coords*/
movtcurabs(&tx, &ty);           /*move text cursor*/
text(icon4);                    /*label variable attribute
                                  element*/

iconcolor=YELLOW;               /*function element is
                                  yellow*/
x1=XLLF; y1=YLLF;               /*lower left coords of icon
                                  area*/
f_icon(x1, y1, iconcolor);      /*draw function element*/

textforeground=BLACK; textbackground=iconcolor;  /*dot text
                                                   colors*/
settextclr(&textforeground, &textbackground);    /*set
                                                   colors*/
tx=x1+0.45; ty=y1+0.2;          /*function element label
                                  coords*/
movtcurabs(&tx, &ty);           /*move text cursor*/
text(icon5);                    /*label function element*/
```

```
    iconcolor=RED;                  /*test element is red*/
    x1=XLLT; y1=YLLT;               /*lower left coords of icon
                                      area*/
    t_icon(x1, y1, iconcolor);      /*draw test element*/

    textforeground=BLACK; textbackground=iconcolor;  /*dot text
                                                       colors*/
    settextclr(&textforeground, &textbackground);    /*set
                                                       colors*/
    tx=x1+0.45; ty=y1+0.2;          /*test element label
                                      coords*/
    movtcurabs(&tx, &ty);           /*move text cursor*/
    text(icon6);                    /*label test element*/

    iconcolor=MAGENTA;              /*module is magenta*/
    x1=XLLM; y1=YLLM;               /*lower left coords of icon
                                      area*/
    m_icon(x1, y1, iconcolor);      /*draw module icon*/

    textforeground=BLACK; textbackground=iconcolor;  /*dot text
                                                      colors*/
    settextclr(&textforeground, &textbackground);    /*set
                                                       colors*/
    tx=x1+0.65; ty=y1+0.25;         /*module label coords*/
    movtcurabs(&tx, &ty);           /*move text cursor*/
    text(icon7);                    /*label module*/

    deltcur();              /*turn off text cursor*/

    textforeground=WHITE; textbackground=BLACK;   /*dot text
                                                    colors*/
    settextclr(&textforeground, &textbackground); /*set colors*/

    x1=0.2; y1=0.35;       /*coords of line 1 of dialog box*/
    movtcurabs(&x1, &y1);  /*move text cursor to line 1*/

}  /*end scrngen*/
```

```
/*****************************************************************
*  Name:     menu.c                                                *
*  Purpose:  to print a menu for user to select the type of      *
*            graphics device installed.                          *
*  Author:   David D. O'Dell                                     *
*  Date:     21 December 1987                                    *
*****************************************************************/



devmenu()
{
     int row, col;            /*location of cursor on text
 screen*/
     int dev;                 /*type of graphics card selected*/

     clearscn();             /*clear the text screen*/
     row=6; col=1;           /*center menu on screen*/
     curlocate(row,col);     /*move cursor*/

     printf
("%63s\n%64s\n%64s\n%64s\n%64s\n%64s\n%64s\n%64s\n%64s\n%64s
      \n%64s",
           "***********************************************",
           "*                                             *",
           "* Select the graphics device for your system: *",
           "*                                             *",
           "*   1.  IBM Color Graphics Adapter            *",
           "*   2.  Generic CGA-type display card         *",
           "*   3.  IBM Enhanced Graphics Adapter         *",
           "*   4.  Sigma Designs Color 400               *",
           "*   5.  Quit                                  *",
           "*                                             *",
           "***********************************************");

     printf("\n\n%28s","Your choice?");

     scanf("%d",&dev);        /*get user's selection*/

     while ((dev<1) || (dev>5))
     {
          printf("Sorry, your selection must be between 1 and
                  5.\n");
          printf("Please re-enter your choice.");

          scanf("%d",&dev);   /*get the new selection*/

     }  /*endwhile*/

     return(dev);
}  /*end devmenu*/
```

```
clearscn()

/*****************************************************************
*  Function:  clear the entire text screen to background color *
*  Modified from blanksc function contained in:                  *
*           Radcliffe, Robert A. and Raab, Thomas J. 1986.       *
*           DATA HANDLING UTILITIES IN C. Berkeley, CA.:         *
*           SYBEX Inc.                                           *
*****************************************************************/

{
     typedef char byte;
     typedef union {int i2; long int i4;} INT;

     struct XREG
     {
          short ax,bx,cx,dx,si,di;
     };

     struct  HREG
     {
          byte  al,ah,bl,bh,cl,ch,dl,dh;
     };

     union REGS
     {
          struct XREG x;
          struct HREG h;
     };

     /*-------------------------------------------------------*/


     union REGS ir, or;
     ir.h.ah = 0x06;    /*interrupt number to scroll page up*/
     ir.h.al = 0;       /*set registers to scroll entire screen*/
     ir.h.ch = 0;       /*from top left (0,0)*/
     ir.h.cl = 0;
     ir.h.dh = 25;            /*to bottom right (25,80)*/
     ir.h.dl = 80;
     ir.h.bh = 0x07;          /*scroll white on black*/
     int86(0x10, &ir, &or);  /*Lattice interrupt call to BIOS
                               CRT*/
}  /*end clearscn*/
```

```
curlocate(row,col)
int row, col;

/*****************************************************************
*  Function:  locate cursor at row and column on text screen     *
*  Modified from setcrc function contained in:                   *
*     Radcliffe, Robert A. and Raab, Thomas J. 1986.             *
*     DATA HANDLING UTILITIES IN C. Berkeley, CA.:  SYBEX Inc.   *
*****************************************************************/


{
     typedef char byte;
     typedef union {int i2; long int i4;} INT;

     struct XREG
     {
          short ax,bx,cx,dx,si,di;
     };

     struct  HREG
     {
          byte  al,ah,bl,bh,cl,ch,dl,dh;
     };

     union REGS
     {
          struct XREG x;
          struct HREG h;
     };


/*---------------------------------------------------------------*/


     union REGS ir, or;
     ir.h.ah = 0x02;    /*interrupt number to set cursor
                             position*/
     ir.h.bh = 0;
     ir.h.dh = row;
     ir.h.dl = col;
     int86(0x10, &ir, &or);  /*Lattice interrupt call to BIOS
                             CRT*/
}  /*end curlocate*/
```

```
/*****************************************************************
*  Name:     icons.c                                              *
*  Purpose:  to create the icons used by Intuition                *
*  Author:   David D. O'Dell                                      *
*  Date:     16 December 1987                                     *
*****************************************************************/


#include "intuit.h"
#include "exstruct.h"

/*all icons are described in terms of world coordinates*/

pe_icon(x, y, iconcolor)   /*create primitive entity icon*/
     float x, y;           /*lower left coord of icon area*/
     int iconcolor;        /*color of icon*/
     {
          float x1, x2, y1, y2;      /*coords of icon*/

          setcolor(&iconcolor);      /*set icon color*/

          x1=x+0.2; y1=y+0.05;       /*lower left coords of
                                       icon*/
          x2=x+0.8; y2=y+0.65;       /*upper right coord of
                                       icon*/
          bar(&x1, &y1, &x2, &y2);   /*draw primitive entity
                                       icon*/

          iconcolor=BLACK;           /*change colors for
                                       outline*/
          setcolor(&iconcolor);      /*set outline color*/

          box(&x1, &y1, &x2, &y2);   /*draw outline*/
     }
/*end pe_icon*/


va_icon(x, y, iconcolor)   /*create compound entity icon*/
     float x, y;           /*lower left coord of icon area*/
     int iconcolor;        /*color of icon*/
     {
          float xarray[8];         /*x-coords of icon*/
          float yarray[8];         /*y-coords of icon*/
          float x1, y1;            /*coords of first vertex*/
          int n;                   /*number of vertices*/

          n=8;                     /*number of vertices*/

         xarray[0]=x+0.6; yarray[0]=y+0.05;  /*coords of
                                               vertices*/
          xarray[1]=x+0.8; yarray[1]=y+0.25;
          xarray[2]=x+0.8; yarray[2]=y+0.45;
```

```
          xarray[3]=x+0.6; yarray[3]=y+0.65;
          xarray[4]=x+0.4; yarray[4]=y+0.65;
          xarray[5]=x+0.2; yarray[5]=y+0.45;
          xarray[6]=x+0.2; yarray[6]=y+0.25;
          xarray[7]=x+0.4; yarray[7]=y+0.05;

          x1=x+0.4; y1=y+0.05;          /*coord of first vertex*/
          movabs(&x1, &y1);             /*move graphics cursor to
                                          vertex*/
          polyfabs(xarray, yarray, &n, &iconcolor);  /*draw
                                                       icon*/

          iconcolor=BLACK;          /*change colors for outline*/
          setcolor(&iconcolor);     /*set outline color*/

          x1=x+0.4; y1=y+0.05;      /*coord of first vertex*/
          movabs(&x1, &y1);         /*move graphics cursor to
                                      vertex*/
          polylnabs(xarray, yarray, &n);  /*draw outline*/
     }
/*end va_icon*/

a_icon(x, y, iconcolor)  /*create attribute element icon*/
     float x, y;         /*lower left coord of icon area*/
     int iconcolor;      /*color of icon*/
     {
          float x1, y1;  /*coords of circle center point*/
          float radius;  /*radius of circle*/

          setcolor(&iconcolor);       /*set icon color*/

          x1=x+0.5; y1=y+0.35;        /*center of circle*/
          movabs(&x1, &y1);           /*move graphics cursor to
                                        center*/
          radius=0.25;                /*set radius of circle*/
          fcir(&radius);              /*draw attribute element
                                        icon*/

          iconcolor=BLACK;            /*change colors for
                                        outline*/
          setcolor(&iconcolor);       /*set outline color*/
          cir(&radius);               /*draw outline*/
     }
/*end a_icon*/

t_icon(x, y, iconcolor)           /*create variable attribute
                                    icon*/
     float x, y;                  /*lower left coord of icon
                                    area*/
     int iconcolor;               /*color of icon*/
     {
          float xarray[4];        /*x-coords of icon*/
```

```
          float yarray[4];        /*y-coords of icon*/
          float x1, y1;           /*coords of first vertex*/
          int n;                  /*number of vertices*/

          n=4;                    /*number of vertices*/
          xarray[0]=x+0.8; yarray[0]=y+0.35;  /*coords of
                                                vertices*/
          xarray[1]=x+0.5; yarray[1]=y+0.65;
          xarray[2]=x+0.2; yarray[2]=y+0.35;
          xarray[3]=x+0.5; yarray[3]=y+0.05;

          x1=x+0.5; y1=y+0.05;        /*coord of first vertex*/
          movabs(&x1, &y1);           /*move graphics cursor to
                                        vertex*/
          polyfabs(xarray, yarray, &n, &iconcolor);  /*draw
                                                       icon*/

          iconcolor=BLACK;          /*change colors for
                                      outline*/
          setcolor(&iconcolor);     /*set outline color*/

          x1=x+0.5; y1=y+0.05;        /*coord of first vertex*/
          movabs(&x1, &y1);           /*move graphics cursor to
                                        vertex*/
          polylnabs(xarray, yarray, &n);  /*draw outline*/
     }
/*end t_icon*/


f_icon(x, y, iconcolor)           /*create function element
                                    icon*/
     float x, y;                  /*lower left coord of icon
                                    area*/
     int iconcolor;               /*color of icon*/
     {
          float xarray[3];        /*x-coords of icon*/
          float yarray[3];        /*y-coords of icon*/
          float x1, y1;           /*coords of first vertex*/
          int n;                  /*number of vertices*/

          n=3;                    /*number of vertices*/

          xarray[0]=x+0.8; yarray[0]=y+0.05;  /*coords of
                                                vertices*/
          xarray[1]=x+0.5; yarray[1]=y+0.65;
          xarray[2]=x+0.2; yarray[2]=y+0.05;

          x1=x+0.2; y1=y+0.05;        /*coord of first vertex*/
          movabs(&x1, &y1);           /*move graphics cursor to
                                        vertex*/
          polyfabs(xarray, yarray, &n, &iconcolor);  /*draw
                                                       icon*/
```

```
          iconcolor=BLACK;           /*change colors for
                                       outline*/
          setcolor(&iconcolor);      /*set outline color*/

          x1=x+0.2; y1=y+0.05;         /*coord of first vertex*/
          movabs(&x1, &y1);            /*move graphics cursor to
                                         vertex*/
          polylnabs(xarray, yarray, &n);  /*draw outline*/
     }
/*end f_icon*/

ce_icon(x, y, iconcolor)             /*create test element icon*/
     float x, y;                     /*lower left coord of icon
                                       area*/
     int iconcolor;                  /*color of icon*/
     {
          float xarray[5];           /*x-coords of icon*/
          float yarray[5];           /*y-coords of icon*/
          float x1, y1;              /*coords of first vertex*/
          int n;                     /*number of vertices*/

          n=5;                       /*number of vertices*/
          xarray[0]=x+0.8; yarray[0]=y+0.05;  /*coords of
                                                vertices*/
          xarray[1]=x+0.8; yarray[1]=y+0.35;
          xarray[2]=x+0.5; yarray[2]=y+0.65;
          xarray[3]=x+0.2; yarray[3]=y+0.35;
          xarray[4]=x+0.2; yarray[4]=y+0.05;

          x1=x+0.2; y1=y+0.05;         /*coord of first vertex*/
          movabs(&x1, &y1);            /*move graphics cursor to
                                         vertex*/
          polyfabs(xarray, yarray, &n, &iconcolor);  /*draw
                                                       icon*/

          iconcolor=BLACK;           /*change colors for
                                       outline*/
          setcolor(&iconcolor);      /*set outline color*/


          x1=x+0.2; y1=y+0.05;         /*coord of first vertex*/
          movabs(&x1, &y1);            /*move graphics cursor to
                                         vertex*/
          polylnabs(xarray, yarray, &n);  /*draw outline*/
     }
/*end ce_icon*/

m_icon(x, y, iconcolor)            /*create module icon*/
     float x, y;                   /*lower left coord of icon
                                     area*/
     int iconcolor;                /*color of icon*/
```

```
    {
        float x1, x2, y1, y2;       /*coords of icon*/

        setcolor(&iconcolor);       /*set icon color*/

        x1=x+0.55; y1=y+0.05;       /*lower left coords of
                                      icon*/
        x2=x+1.45; y2=y+0.65;       /*upper right coord of
                                      icon*/
        bar(&x1, &y1, &x2, &y2);    /*draw primitive entity
                                      icon*/

        iconcolor=BLACK;            /*change colors for
                                      outline*/
        setcolor(&iconcolor);       /*set outline color*/

        box(&x1, &y1, &x2, &y2);    /*draw outline*/

        x1=x+0.9; y1=y+0.05;        /*set start of cross line*/
        movabs(&x1, &y1);           /*move graphics cursor to
                                      start*/
        x2=x+0.9; y2=y+0.65;        /*set end of cross line*/
        lnabs(&x2, &y2);            /*draw cross line*/

    }
/*end m_icon*/


arrow(arrowcolor)               /*create directed arc arrow head*/
    int arrowcolor;
    {
        int hcolor;                 /*color of h-cursor*/
        float cx, cy;               /*coords of node ctr*/
        float x, y;                 /*coords of arrow tip*/
        float xarray[3];            /*x-coords of arrow*/
        float yarray[3];            /*y-coords of arrow*/
        float x1, y1;               /*coords of stem end*/
        int n;                      /*number of vertices*/

        n=3;                        /*number of vertices*/

        inqhcur(&cx,&cy,&hcolor);

        y=cy-0.55;                      /*set tip y-coord*/

        if(cell[active_node].out==1)  /*set tip x-coord*/
            x=cx;
        else if(cell[active_node].out==2)
            x=cx-0.2;
        else if(cell[active_node].out==3)
            x=cx+0.2;
        else if(cell[active_node].out==4)
```

```
            x=cx-0.4;
        else if(cell[active_node].out==5)
            x=cx+0.4;
        else
        {
            x=cx;  /*default*/
        }

        setcolor(&arrowcolor);

        xarray[0]=x-0.05; yarray[0]=y-0.1;   /*coords of
                                               vertices*/
        xarray[1]=x+0.05; yarray[1]=y-0.1;
        xarray[2]=x    ; yarray[2]=y;

        movabs(&x, &y);     /*move graphics cursor to tip*/
        polyfabs(xarray, yarray, &n, &arrowcolor);  /*draw
                                                      arrow*/

        x1=x; y1=y-0.2;
        lnabs(&x1,&y1);            /*draw stem*/

    }
/*end arrow*/
```

```
/*************************************************************
*  Name:     util.c                                          *
*  Purpose:  contains general purpose utility subroutines    *
*            used by various programs.                       *
*  Author:   David D. O'Dell                                 *
*  Date:     31 December 1987                                *
*************************************************************/

#include "intuit.h"
#include "exstruct.h"


bigport()

/*************************************************************
*  Function:  set viewport to full screen normalized        *
*             coordinates.  Do not draw border or redraw    *
*             background                                      *
*************************************************************/

{
     float x1, y1, x2, y2;              /*graphic coordinate
                                          variables*/
     int border;                        /*flag to draw viewport
                                          border*/
     int background;                    /*flag for viewport
                                          background*/

     x1=0.0; y1=0.0; x2=1.0; y2=1.0;    /*full screen normalized
                                          coords*/
     border=-1;background=-1;           /*no border or
                                          background*/
     setviewport(&x1, &y1, &x2, &y2,&border,&background);  /*set
                                                    viewport*/

     x1=WXMIN; y1=WYMIN; x2=WXMAX; y2=WYMAX;
     setworld(&x1, &y1, &x2, &y2);      /*set world coord system*/

} /*endbigport*/


littleport()

/*************************************************************
*  Function:  set viewport to work area only in normalized *
*             coordinates.  Do not draw border or redraw   * *
         background                                          *
*************************************************************/

{
     float x1, y1, x2, y2;        /*graphic coordinate variables*/
```

```
     float xw, yw;                /*float to normal coords*/
     int border;                  /*flag to draw viewport border*/

     int background;              /*flag for viewport background*/

     int height, width, path, textmode; /*dot text attributes*/

     xw=XLLWORK;yw=YLLWORK;       /*set lower lt corner of
                                    viewport*/
     mapwton(&xw,&yw,&x1,&y1);    /*to lower lt corner of work
                                    area*/
     xw=XURWORK;yw=YURWORK;       /*set upper rt corner of
                                    viewport*/
     mapwton(&xw,&yw,&x2,&y2);    /*to upper rt corner of work
                                    area*/
     border=-1;background=-1;     /*no border or background*/
     setviewport(&x1,&y1,&x2,&y2,&border,&background);  /*set
                                                    viewport*/

     x1=WXMIN; y1=WYMIN; x2=WXMAX; y2=WYMAX+2.0;
     setworld(&x1, &y1, &x2, &y2);     /*set world coord system*/


     height=TEXTHT; width=TEXTWD; path=HORIZONTAL;
textmode=BORDER;
     settext(&height,&width,&path,&textmode);

}  /*endlittleport*/



char getchr(kbchar)
char *kbchar;

/*************************************************************
*  Function:  get character from keyboard without echo and   *
*             return character by function and value.        *
*  Modified from getchr.c contained in:                       *
*           Radcliffe, Robert A. and Raab, Thomas J. 1986.    *
*           DATA HANDLING UTILITIES IN C. Berkeley, CA.:      *
*           SYBEX Inc.                                        *
*************************************************************/

{
     typedef char byte;
     typedef union {int i2; long int i4;} INT;

     struct XREG
     {
          short ax,bx,cx,dx,si,di;
     };
```

```
     struct  HREG
     {
          byte  al,ah,bl,bh,cl,ch,dl,dh;
     };

     union REGS
     {
          struct XREG x;
          struct HREG h;
     };


/*-------------------------------------------------------*/

     union REGS ir, or;
     ir.h.ah = 0x07;          /*get character for keyboard*/
     int86(0x21, &ir, &or);   /*DOS function call access*/
     *kbchar = or.h.al;       /*put character in kbchar*/
     return(or.h.al);
}  /*end getchr*/



decode_char(sch)
char sch;

/*****************************************************************
*  Function:  check if a cursor key is pressed and update      *
*             cx and cy coordinates accordingly                *
*****************************************************************/

{
     float top_edge, bottom_edge;  /*top and bottom cursor
                                     limits*/
     float left_edge, right_edge;  /*left and right cursor
                                     limits*/
     float button_height, button_width;
     float hheight, hwidth;
     static float cx,cy;           /*local cursor variables*/
     int on;                       /*button state*/
     int hcolor;                   /*x-hair color*/

     on=0;                         /*no button selected*/

     button_height=YURF1-YLLF1;
     button_width=XURF1-XLLF1;
     hheight=button_height/2.0;
     hwidth=button_width/2.0;

     inqhcur(&cx,&cy,&hcolor);     /*get postition of x-hair
                                     cursor*/
```

```
if((sch==72)||(sch==75)||(sch==77)||(sch==80))
{
     top_edge = YURBUTTONS-hheight;
     bottom_edge = YLLBUTTONS+hheight;
     left_edge = XLLBUTTONS+hwidth;
     right_edge = XURBUTTONS-hwidth;

     switch (sch)
     {
          case 72:  cy=cy+(2*hheight);  /*cursor up height
                                          x_hair*/
                    if(cy>top_edge)     /*if at top edge*/
                         cy=top_edge;   /*stay there*/
                    break;
          case 75:  cx=cx-(2*hwidth);   /*cursor left width
                                          x-hair*/
                    if(cx<left_edge)    /*if at left edge*/
                         cx=left_edge;  /*stay there*/
                    break;
          case 77:  cx=cx+(2*hwidth);   /*cursor right width
                                          x-hair*/
                    if(cx>right_edge)   /*if at right edge*/

                         cx=right_edge; /*stay there*/
                    break;
          case 80:  cy=cy-(2*hheight);   /*cursor down
                                           height x-hair*/
                    if(cy<bottom_edge)   /*if at bottom
                                           edge*/
                         cy=bottom_edge; /*stay there*/
                    break;
          default:  break;

     }  /*endswitch*/

     movhcurabs(&cx,&cy);
     delhcur();                /*don't show x-hair cursor*/

     if(inside(cx,cy,XLLF1,YLLF1,XURF1,XURF1))
          on=1;
     else if(inside(cx,cy,XLLF2,YLLF2,XURF2,XURF2))
          on=2;
     else if(inside(cx,cy,XLLF3,YLLF3,XURF3,XURF3))
          on=3;
     else if(inside(cx,cy,XLLF4,YLLF4,XURF4,XURF4))
          on=4;
     else if(inside(cx,cy,XLLF5,YLLF5,XURF5,XURF5))
          on=5;
     else if(inside(cx,cy,XLLF6,YLLF6,XURF6,XURF6))
          on=6;
     else if(inside(cx,cy,XLLF7,YLLF7,XURF7,XURF7))
          on=7;
```

```
          else if(inside(cx,cy,XLLF8,YLLF8,XURF8,XURF8))
               on=8;
          else if(inside(cx,cy,XLLF9,YLLF9,XURF9,XURF9))
               on=9;
          else if(inside(cx,cy,XLLF10,YLLF10,XURF10,XURF10))
               on=10;
          else
               on=0;

     }  /*end if sch*/

/*************************************************************
*  Function:  check if a function key has been pressed      *
*             and return the corresponding value of 'on'.   *
*************************************************************/

     if((sch>=59)&&(sch<=68))
     {
          switch(sch)
          {
               case 59:  on=1;
                         cx=XLLF1+hwidth;   /*position x-hair
                                              cursor*/
                         cy=YLLF1+hheight;  /*in F1 button*/
                         break;
               case 60:  on=2;
                         cx=XLLF2+hwidth;   /*position x-hair
                                              cursor*/
                         cy=YLLF2+hheight;  /*in F2 button*/
                         break;
               case 61:  on=3;
                         cx=XLLF3+hwidth;   /*position x-hair
                                              cursor*/
                         cy=YLLF3+hheight;  /*in F3 button*/
                         break;
               case 62:  on=4;
                         cx=XLLF4+hwidth;   /*position x-hair
                                              cursor*/
                         cy=YLLF4+hheight;  /*in F4 button*/
                         break;
               case 63:  on=5;
                         cx=XLLF5+hwidth;   /*position x-hair
                                              cursor*/
                         cy=YLLF5+hheight;  /*in F5 button*/
                         break;
               case 64:  on=6;
                         cx=XLLF6+hwidth;   /*position x-hair
                                              cursor*/
                         cy=YLLF6+hheight;  /*in F6 button*/
                         break;
```

```
                case 65:  on=7;
                          cx=XLLF7+hwidth;    /*position x-hair
                                                cursor*/
                          cy=YLLF7+hheight;   /*in F7 button*/
                          break;
                case 66:  on=8;
                          cx=XLLF8+hwidth;    /*position x-hair
                                                cursor*/
                          cy=YLLF8+hheight;   /*in F8 button*/
                          break;
                case 67:  on=9;
                          cx=XLLF9+hwidth;    /*position x-hair
                                                cursor*/
                          cy=YLLF9+hheight;   /*in F9 button*/
                          break;
                case 68:  on=10;
                          cx=XLLF10+hwidth;    /*position x-hair
                                                 cursor*/
                          cy=YLLF10+hheight;  /*in F10 button*/
                          break;
                default:  on=0;
                          break;
           }  /*end switch*/

           movhcurabs(&cx,&cy);
           delhcur();

      }  /*end if sch*/

      return(on);

}  /*end decode_char*/



int inside(x,y,xmin,ymin,xmax,ymax)
float x,y;                          /*location of cursor*/
float xmin,ymin,xmax,ymax;          /*boundries of bounding box*/

/*************************************************************
*  Function:  determines if (x,y) is inside bounding box    * *
         determined by  (xmin,ymin)-(xmax,ymax).       *
*************************************************************/
{
      if((xmin <= x) && (x <= xmax) && (ymin <= y) && (y <= ymax))

      {
           return(TRUE);   /*cursor is inside bounding box*/
      }
      else
      {
           return(FALSE); /*cursor is outside bounding box*/
```

```
     }  /*endif-else*/
}  /*end inside*/


draw_submenu(title,line1,line2,line3,line4,line5,menu)
char *title;
char *line1,*line2,*line3,*line4,*line5;
int *menu;
/*************************************************************
*  Function:  prints submenu for selected.                   *
*************************************************************/
{
     int textforeground, textbackground;  /*dot text
                                            colors*/
     int color;                           /*drawing color*/
     float ulx, uly, lrx, lry;            /*coord of popup menu
                                            area*/
     float sx1, sx2, sy1, sy2;            /*graphic coordinates*/

     float stx, sty;               /*dot text coordinates*/
     char getchr();                /*get keyboard character*/

     define_dot_text();            /*set dot text attributes*/
     textforeground=WHITE;
     textbackground=BLACK;
     settextclr(&textforeground,&textbackground);

     ulx=1.0; uly=4.1;             /*upper left coord of popup*/
     lrx=5.0; lry=2.0;             /*lower right coord of popup*/

     movefrom(&ulx,&uly,&lrx,&lry,menu);  /*save scrn under
                                            popup*/

     color=BLACK;
     setcolor(&color);
     sx1=1.0; sy1=2.0;
     sx2=5.0; sy2=4.1;
     bar(&sx1,&sy1,&sx2,&sy2);     /*draw popup menu*/
     color=WHITE;
     setcolor(&color);
     sx1=1.1; sy1=2.1;
     sx2=4.9; sy2=3.6;
     box(&sx1,&sy1,&sx2,&sy2);
     sx1=1.1; sy1=3.7;
     sx2=4.9; sy2=4.0;
     box(&sx1,&sy1,&sx2,&sy2);

     stx=2.5; sty=3.75;
     movtcurabs(&stx,&sty);
     text(title);

     stx=1.5; sty=3.35;
```

```
      movtcurabs(&stx,&sty);
      text(line1);

      stx=1.5; sty=3.05;
      movtcurabs(&stx,&sty);
      text(line2);

      stx=1.5; sty=2.75;
      movtcurabs(&stx,&sty);
      text(line3);

      stx=1.5; sty=2.45;
      movtcurabs(&stx,&sty);
      text(line4);

      stx=1.5; sty=2.15;
      movtcurabs(&stx,&sty);
      text(line5);

      deltcur();

}  /*end draw submenu*/


restore_under_menu(menu)
int *menu;
/*************************************************************
*  Function:  restore screen area under submenu.             *
*************************************************************/
{
     float ulx, uly, lrx, lry;        /*popup menu coordinates*/
     float sx1, sy1, sx2, sy2;        /*graphics coordinates*/
     int mode;                        /*restore mode*/
     int color;                       /*current drawing color*/

     color=WHITE;
     setcolor(&color);
     sx1=1.0; sy1=2.0;
     sx2=5.0; sy2=4.0;
     bar(&sx1,&sy1,&sx2,&sy2);     /*draw popup menu*/

     ulx=1.0; uly=4.1;             /*upper left coord of popup*/
     lrx=5.0; lry=2.0;             /*lower right coord of popup*/


     mode=1;                       /*set restore to overwrite*/
     moveto(&ulx,&uly,menu,&mode); /*restore scrn under popup*/

}  /*end restore_under_menu*/
```

```
define_dot_text()
/*************************************************************
*  Function:  sets dot text attributes.                      *
*************************************************************/
{
     int device;                          /*graphics device*/
     int height, width, path, textmode;  /*dot text attributes*/

     inqdev(&device);          /*get device number installed*/

     if((device==1)||(device==26))
          height=TEXTHT;
     else
          height=(2*TEXTHT);

     width=TEXTWD;
     path=HORIZONTAL;
     textmode=BORDER;
     settext(&height,&width,&path,&textmode);

}  /*define dot text*/



write_error(error_number)
/*************************************************************
*  Function:  prints error message pointed to by error number *
*************************************************************/
{
     int dev, device;                    /*graphics device*/
     int textforeground,textbackground; /*dot text colors*/
     float stx, sty;               /*dot text coordinates*/

     static char error1[]=
               "Sorry, selection out of range. Please re-enter
                choice. ";

     define_dot_text();        /*set dot text attributes*/

     dev=inqdev(&device);      /*get graphics device number*/
     if((dev==1)||(dev==26))
          textforeground=WHITE;
     else
          textforeground=RED;
     textbackground=BLACK;
     settextclr(&textforeground,&textbackground);

     clear_dialog_box();

     stx=0.2; sty=0.35;
     movtcurabs(&stx,&sty);
```

```
     switch(error_number)
     {
          case 1:  text(error1);
                   break;
          default: break;
     }  /*end switch*/

}  /*end write error*/



clear_dialog_box()
/*************************************************************
*  Function:  clear error statement from dialog box.          *
*************************************************************/
{
     int color;                          /*drawing color*/
     float x1, y1, x2, y2;               /*graphics coordinates*/

     color=BLACK;                  /*set drawing color to black*/
     setcolor(&color);

     x1=XLLDIALOG+0.045; y1=YLLDIALOG+0.045;
     x2=XURDIALOG-0.055; y2=YURDIALOG-0.055;
     bar(&x1,&y1,&x2,&y2);               /*overwrite dialog box*/

     color=WHITE;
     setcolor(&color);

}  /*end clear dialog box*/



clear_status_box()
/*************************************************************
*  Function:  clear status box of all text.                   *
*************************************************************/
{
     int color;                          /*drawing color*/
     float x1, y1, x2, y2;               /*graphics coordinates*/

     color=BLACK;                  /*set drawing color to black*/
     setcolor(&color);

     x1=XLLSTATUS+0.8; y1=YLLSTATUS+0.05;
     x2=XURSTATUS-2.2; y2=YURSTATUS-0.05;
     bar(&x1,&y1,&x2,&y2);                    /*overwrite type*/

     x1=XLLSTATUS+4.6; y1=YLLSTATUS+0.05;
     x2=XURSTATUS-0.5; y2=YURSTATUS-0.05;
     bar(&x1,&y1,&x2,&y2);                    /*overwrite name*/
}  /*end clear status box*/
```

```
write_dialog(message1,message2)
char *message1, *message2;
/*****************************************************************
*  Function:  write two line message in dialog box at bottom of *
*             screen.                                            *
*****************************************************************/

{
     int textforeground, textbackground;      /*dot text colors*/

     float tx, ty;                          /*dot text coordinates*/

     define_dot_text();                /*set dot text attributes*/


     textforeground=WHITE;
     textbackground=BLACK;
     settextclr(&textforeground,&textbackground);

     clear_dialog_box();

     tx=XLLDIALOG+0.2; ty=YLLDIALOG+0.3;   /*write first line*/
     movtcurabs(&tx,&ty);
     text(message1);

     tx=XLLDIALOG+0.2; ty=YLLDIALOG+0.05;  /*write second line*/
     movtcurabs(&tx,&ty);
     text(message2);

}  /*end write dialog*/



getstring(string,length,tx,ty)
char *string;
int length;
float *tx, *ty;
/*****************************************************************
*  Function:  get and print a string at specified location.  *
*****************************************************************/
{
     int textforeground, textbackground;  /*dot text colors*/
     int index;
     char ch;
     float x,y;                         /*dot text coordinates*/

     index=0;
     x=*tx; y=*ty;

     textforeground=CYAN;
     textbackground=BLACK;
     settextclr(&textforeground,&textbackground);
```

```
    getchr(&ch);

    while(!(ch==0x0D))          /*while not carriage return*/
    {
         if(ch==0x0D)
         {
              ch='\0';
              string[index]=ch;
              break;
         }
         else if(ch==0x08)
         {
              textforeground=BLACK;textbackground=BLACK;
              settextclr(&textforeground,&textbackground);
              movtcurabs(&x,&y);
              text(string);
              --index;
              if(index<0)
                   index=0;
              ch=' ';
              string[index]=ch;
              textforeground=CYAN;
              textbackground=BLACK;
              settextclr(&textforeground,&textbackground);
              movtcurabs(&x,&y);
              text(string);
         }
         else if((ch>='0')&&(ch<='9')||
                 (ch>='A')&&(ch<='Z')||
                 (ch>='a')&&(ch<='z')||
                 (ch='_'))
         {
              if(index>length-1)
              {
                   index=length;
                   string[index]='\0';
                   break;
              }

              string[index]=ch;
              movtcurabs(&x,&y);
              text(string);

              ++index;

         }  /*end if-else*/

         getchr(&ch);

    }  /*end while*/
}  /*end getstring*/
```

```
rest_under_screen(menu)
int *menu;
/*************************************************************
*  Function:  restore screen area under edit screen.        *
*************************************************************/
{
     float ulx, uly;                    /*popup menu coordinates*/
     float sx1, sy1, sx2, sy2;          /*graphics coordinates*/
     int mode;                          /*restore mode*/
     int color;                         /*current drawing color*/

     color=WHITE;
     setcolor(&color);
     sx1=0.5; sy1=1.0;
     sx2=5.5; sy2=4.5;
     bar(&sx1,&sy1,&sx2,&sy2);      /*draw popup menu*/

     ulx=0.5; uly=4.5;              /*upper left coord of popup*/

     mode=1;                        /*set restore to overwrite*/
     moveto(&ulx,&uly,menu,&mode); /*restore scrn under popup*/

}  /*end restore_under_screen*/



get_name()
/*************************************************************
*  Function:  gets element name and writes it below icon    *
*             image in work screen.                         *
*************************************************************/
{
     static char name[8];               /*8 characters + null*/
     static char message1[]="MANDATORY ENTRY:  Enter element
                            name";
     static char message2[]="(7 or less characters) and press
                            RETURN";
     static char message3[]=
     "Move cursor and press p,c,a,v,f,t, or m to add an";
     static char message4[]=
     "element or press e to add edge or <esc> to quit.";

     int i;                             /*float variable*/
     int hcurclr;                       /*x-hair cursor color*/
     int strlength;                     /*# of characters in name*/

     float cx, cy;                     /*pos of x-hair box cursor*/


     strlength=7;

     for(i=0; i<=(strlength-2); i++)      /*init name to blanks*/
```

```
        name[i]=' ';
    name[strlength-1]='\0';

    inqhcur(&cx,&cy,&hcurclr);
    cx=cx-0.45; cy=cy-0.45;        /*put name in bottom of box*/


    bigport();
    write_dialog(&message1,&message2);
    littleport();

    getstring(&name,strlength,&cx,&cy);            /*get name*/

    bigport();
    write_dialog(&message3,&message4);
    littleport();

    element[active_node].ename=name;
    cell[active_node].ename=name;

}  /*end get name*/



get_model_name()
/*************************************************************
*  Function:  get name of model user will enter.             *
*************************************************************/
{
    int i;                                  /*loop variable*/
    int tcurclr;                            /*text cursor color*/
    int textforeground, textbackground;     /*dot text colors*/
    int strlength;                          /*string length*/
    float tx, ty;      /*dot text coordinates*/
    float dtx, dty;    /*position of text cursor in dialog box*/

    static char name[11];
    static char message1[]="Enter the name of this model";
    static char message2[]="(10 or less characters):    ";

    strlength=10;

    clear_status_box();           /*clear model type and name*/

    for(i=0;i<=(strlength-2);i++) /*init model name to blanks*/
        name[i]=' ';
    name[strlength-1]='\0';

    define_dot_text();

    textforeground=YELLOW;
```

```
        textbackground=BLACK;
        settextclr(&textforeground,&textbackground);

        clear_dialog_box();
        write_dialog(&message1,&message2);

        inqtcur(&dtx, &dty, &tcurclr);  /*get text cursor position*/

        getstring(&name,strlength,&dtx,&dty);

        textforeground=YELLOW;
        textbackground=BLACK;
        settextclr(&textforeground,&textbackground);
        tx=XLLSTATUS+4.7; ty=YLLSTATUS+0.05;
        movtcurabs(&tx,&ty);
        text(name);

        model_name=name;

        deltcur();                          /*turn off text cursor*/
        clear_dialog_box();

}  /*end get model name*/


init_workspace()
{
        int i,j,k;                              /*loop variables*/

        for(i=0; i<=MAXNODES; i=i+1)
        {
             element[i].ename="null";
             element[i].etype="null";
             element[i].dname="null";
             element[i].date_added="null";
             element[i].last_mod="null";
             element[i].nmods=0;
             element[i].idx="null";
             element[i].idx_stmt="null";
             element[i].grange="null";
             element[i].grule="null";
             element[i].comments="null";
             element[i].rtype="null";
             element[i].acc_meth="null";
             element[i].freq=0;
             element[i].rel_pos=0;
        }  /*end for*/

        for(i=0; i<=MAXNODES; i=i+1)
        {
             cell[i].used=FALSE;
             cell[i].ename="null";
```

```
            cell[i].nodex=0.0;
            cell[i].nodey=0.0;
            cell[i].out=0;
            cell[i].in=0;

            for(j=0; j<=MAXEDGES; j=j+1)
            {
                 for(k=0; k<=MAXPOINTS; k=k+1)
                 {
                      cell[i].edgex[j][k]=0.0;
                      cell[i].edgey[j][k]=0.0;
                 }  /*end for k*/

                 cell[i].cnode[j]=0;
                 cell[i].cedge[j]=0;
                 element[i].e1name[j]="null";
                 element[i].e1type[j]="null";
                 element[i].e2name[j]="null";
                 element[i].e2type[j]="null";
            }  /*end for j*/
       }  /*end for i*/

       model_name="null";
       used_nodes=0;
       active_node=0;

       littleport();
       clr();
       bigport();

}  /*end init_workspace*/
```

```
/*****************************************************************
*  Name:     fun.c                                                *
*  Purpose:  contains functions assigned to function keys.        *
*  Author:   David D. O'Dell                                      *
*  Date:     29 December 1987                                     *
*****************************************************************/

#include "intuit.h "
#include "exstruct.h"

     int menu[12000];          /*global array to hold screen area
                                 under popup menus*/

f1mode()
/*****************************************************************
*  Function:  allows user to select between creating a genus *
*             graph, creating a module tree, or editing model*
*             paragraphs.                                        *
*****************************************************************/
{
     char ch, sch;                        /*keyboard character*/

     /*submenu title entries*/
     static char subtitle[]=" F1 MODE ";
     static char subline1[]="1.  Create genus graph       ";
     static char subline2[]="2.  Create module tree       ";
     static char subline3[]="3.  Edit model paragraphs    ";
     static char subline4[]=" Enter your choice or...     ";
     static char subline5[]=" Select a new command button.";

     int on;

     draw_submenu(&subtitle,&subline1,&subline2,
                  &subline3,&subline4,&subline5,&menu);

     getchr(&ch);

     if(ch==0x00)
     {
          getchr(&sch);
          on=decode_char(sch);
          clear_dialog_box();
          restore_under_menu(&menu);
          return(on);
     }
     else if((ch<'1')||(ch>'3'))
          write_error(1);
     else if(ch=='1')
     {
          restore_under_menu(&menu);
          clear_dialog_box();
          create_genus_graph();
```

```
     }
     else if(ch=='2')
     {
          restore_under_menu(&menu);
          clear_dialog_box();
          create_module_tree();
     }
     else if(ch=='3')
     {
          clear_dialog_box();
          restore_under_menu(&menu);
          edit_model_paragraphs();
     }  /*end if-else*/

     on=1;
     return(on);

}  /*end f1mode*/


f2add()
/*****************************************************************
*  Function:  allows user to add nodes or edges to a genus      *
*             graph or module tree interactively in the work    *
*             area.                                             *
*****************************************************************/
{
     char ch, sch;                        /*keyboard characters*/

     /*submenu title entries*/
     static char subtitle[]=" F2 ADD  ";
     static char subline1[]="1.  Add a node                ";
     static char subline2[]="2.  Add an edge               ";
     static char subline3[]="3.  Add a module              ";
     static char subline4[]="                              ";
     static char subline5[]="Enter selection...            ";

     int on;

     draw_submenu(&subtitle,&subline1,&subline2,
                  &subline3,&subline4,&subline5,&menu);

     getchr(&ch);

     if(ch==0x00)
     {
          getchr(&sch);
          on=decode_char(sch);
     }  /*end if 0x00*/

     clear_dialog_box();    /*erase any error messages*
```

```
     restore_under_menu(&menu);

     if(ch=='1')
     {
          on=2;
          add_node();
     }
     else if(ch=='2')
     {
          on=2;
          add_edge();
     }
     else if(ch=='3')
     {
          on=2;
          add_module();
     }  /*end if-else*/

     return(on);

}  /*end f2add*/


f3delete()
/*********************************************************************
*  Function:  allows user to delete nodes or edges from a genus *
*             graph or module tree.                                  *
*********************************************************************/
{
     char ch, sch;                          /*keyboard characters*/

     /*submenu title entries*/
     static char subtitle[]="F3 DELETE";
     static char subline1[]="1.  Delete a node            ";
     static char subline2[]="2.  Delete an edge           ";
     static char subline3[]="3.  Delete a module          ";
     static char subline4[]="                             ";
     static char subline5[]="Enter selection...           ";

     int on;

     draw_submenu(&subtitle,&subline1,&subline2,
                  &subline3,&subline4,&subline5,&menu);

     getchr(&ch);

     if(ch==0x00)
     {
          getchr(&sch);
          on=decode_char(sch);
     }  /*end if 0x00*/
```

```
     clear_dialog_box();     /*erase any error messages*/

     restore_under_menu(&menu);

     if(ch=='1')
     {
          on=3;
          delete_node();
     }
     else if(ch=='2')
     {
          on=3;
          delete_edge();
     }
     else if(ch=='3')
     {
          on=3;
          delete_module();
     }  /*end if-else*/

     return(on);

}  /*end f3delete*/


f4change()
/*************************************************************
*  Function:  allows user to change a node type or name in a *
*             genus graph or modular structure.              *
*************************************************************/
{
     char ch, sch;                    /*keyboard characters*/

     /*submenu title entries*/
     static char subtitle[]="F4 CHANGE";
     static char subline1[]="1.  Change element name      ";
     static char subline2[]="2.  Change element type      ";
     static char subline3[]="                             ";
     static char subline4[]="                             ";
     static char subline5[]="Enter selection...           ";
     int on;

     draw_submenu(&subtitle,&subline1,&subline2,
                  &subline3,&subline4,&subline5,&menu);

     getchr(&ch);

     if(ch==0x00)
     {
          getchr(&sch);
          on=decode_char(sch);
     }  /*end if 0x00*/
```

```
     clear_dialog_box();   /*erase any error messages*/

     restore_under_menu(&menu);

     if(ch=='1')
     {
          on=4;
          change_name();
     }
     else if(ch=='2')
     {
          on=4;
          change_type();
     }  /*end if-else*/

     return(on);

}  /*end f4change*/


f5find()
/*****************************************************************
*  Function:  allows user to locate a node in a genus graph or*
*             modular structure.                                  *
*****************************************************************/
{
     char ch, sch;                    /*keyboard characters*/
     int on;

     static char message1[]="FUNCTION NOT IMPLEMENTED.";
     static char message2[]="Select any command to continue.";

     write_dialog(&message1,&message2);

     getchr(&ch);

     if(ch==0x00)
     {
          getchr(&sch);
          on=decode_char(sch);
     }  /*end if 0x00*/

     clear_dialog_box();

     return(on);

}  /*end f5find*/
```

```
f6move()
/*****************************************************************
*  Function:  allows user to re-locate node(s) in a genus graph*
*             or modular structure.                              *
*****************************************************************/
{
     char ch, sch;          /*keyboard characters*/
     int on;
     static char message1[]="FUNCTION NOT IMPLEMENTED.";
     static char message2[]="Select any command to continue.";

     write_dialog(&message1,&message2);

     getchr(&ch);

     if(ch==0x00)
     {
          getchr(&sch);
          on=decode_char(sch);
     }  /*end if 0x00*/

     clear_dialog_box();
     return(on);

}  /*end f6move*/


f7load()
/*****************************************************************
*  Function:  allows user to load a structured model from disk.*
*************************************************** *****/
{
     char ch, sch;          /*keyboard characters*/
     int on;
     static char message1[]="FUNCTION NOT IMPLEMENTED.";
     static char message2[]="Select any command to continue.";

     write_dialog(&message1,&message2);

     getchr(&ch);

     if(ch==0x00)
     {
          getchr(&sch);
          on=decode_char(sch);
     }  /*end if 0x00*/

     clear_dialog_box();

     return(on);

}  /*end f7load*/
```

```
f8save()
/*****************************************************************
*  Function:  allows user to save a structured model to disk.  *
*****************************************************************/
{
     char ch, sch;          /*keyboard characters*/
     int on;
     static char message1[]="FUNCTION NOT IMPLEMENTED.";
     static char message2[]="Select any command to continue.";

     write_dialog(&message1,&message2);

     getchr(&ch);

     if(ch==0x00)
     {
          getchr(&sch);
          on=decode_char(sch);
     }  /*end if 0x00*/

     clear_dialog_box();

     return(on);

}  /*end f8save*/


f9dbms()
/*****************************************************************
*  Function:  allows user to connect to a database management  *
*             system.                                           *
*****************************************************************/
{
     static char message1[]="DBMS function not implemented.";
     static char message2[]="Select any command to continue.";
     char ch, sch;
     int on;

     write_dialog(&message1,&message2);

     getchr(&ch);
     if(ch==0x00)
     {
          getchr(&sch);
          on=decode_char(sch);
     }  /*end if 0x00*/

     clear_dialog_box();

     return(on);

}  /*end f9dbms*/
```

```
f10quit()
/*****************************************************************
*  Function:  allows user to the HALO graphics environment.     *
*****************************************************************/

{
     char ch, sch;                          /*keyboard characters*/

     /*submenu title entries*/
     static char subtitle[]="F10 QUIT ";
     static char subline1[]="1.  Yes, I want to quit.    ";
     static char subline2[]="                            ";
     static char subline3[]="Enter 1 to confirm quitting  ";
     static char subline4[]="         ...or...           ";
     static char subline5[]="Select a new command button. ";

    int on;                             /*selected command button*/

     draw_submenu(&subtitle,&subline1,&subline2,
                  &subline3,&subline4,&subline5,&menu);

     getchr(&ch);

     if(ch==0x00)
     {
          getchr(&sch);
          on=decode_char(sch);
     }  /*end if 0x00*/

     clear_dialog_box();    /*erase any error messages*/

     restore_under_menu(&menu);

     if(ch=='1')
     {
         on=11;
         return(on);
     }

     return(on);

}  /*end f10quit*/
```

```
/*****************************************************************
*  Name:     sfun1.c                                              *
*  Purpose:  contains sub-functions called by primary command     *
*            functions constained in file fun.c                   *
*  Author:   David D. O'Dell                                      *
*  Date:     7 January 1988                                       *
*****************************************************************/


#include "intuit.h"
#include "exstruct.h"

create_genus_graph()
/*****************************************************************
*  Function:  set up screen so user can enter a genus graph.  *
*****************************************************************/
{
     char ch, sch;
     int textforeground, textbackground;   /*dot text colors*/
     float tx, ty;                        /*dot text coordinates*/
     static char message1[]="WARNING:  You will erase the current
                             model!";
     static char message2[]="Press y to erase or <esc> to
                             return.";
     static char graphtype[]="Genus Graph     ";
     static char message3[]="Model initialized and workspace
                             cleared.";
     static char message4[]="Select any command to continue.";

     write_dialog(&message1,&message2);

     define_dot_text();

     textforeground=YELLOW;
     textbackground=BLACK;
     settextclr(&textforeground,&textbackground);

     tx=XLLSTATUS+0.9; ty=YLLSTATUS+0.05;
     movtcurabs(&tx,&ty);
     text(graphtype);

     deltcur();

     getchr(&ch);
     if(ch==0x00)
          getchr(&sch);

     if(ch=='y')
     {
          init_workspace();
          get_model_name();
          write_dialog(&message3,&message4);
```

```
    }
    else
    {
        clear_dialog_box();
        return;
    }  /*end if else*/

}  /*end create genus graph*/


add_node()
{
    float x1, y1, x2, y2;        /*graphic coordinates*/
    float tx, ty;                /*dot text coordinates*/
    float ix, iy;                /*icon area coordinates*/
    float bx1, by1, bx2, by2;    /*cell box coordinates*/
    float hx, hy;                /*x-hair cursor coordinates*/
    float hheight, hwidth;       /*x-hair cursor height/width*/
    float top_edge,bottom_edge;
    float left_edge,right_edge;
    int i;                              /*loop variable*/
    int hcolor;                         /*x-hair cursor color*/
    int textforeground, textbackground; /*dot text color*/
    int iconcolor;
    int color;                  /*current drawing color*/
    char ch, sch;               /*keyboard characters*/
    char row[2], col[2];        /*position of box cursor*/
    static char message1[]=
    "Move cursor and press p, c, a, v, f, t, or m to add an";
    static char message2[]=
    "element or press e to add an edge or <esc> to return.";

    write_dialog(&message1,&message2);

    row[0]='7'; col[0]='A';
    row[1]='\0';col[1]='\0';
    tx=XLLROW+0.8; ty=YLLROW+0.1;
    movtcurabs(&tx, &ty);
    text(row);
    tx=XLLCOL+0.8; ty=YLLCOL+0.1;
    movtcurabs(&tx, &ty);
    text(col);

    littleport();                       /*reset viewport area*/

    hheight=0.5; hwidth=0.5;
    hcolor=BLACK;
    inithcur(&hheight,&hwidth,&hcolor);

    textforeground=WHITE; textbackground=BLACK;
    settextclr(&textforeground, &textbackground);
```

```
hx=WXMIN+0.5; hy=WYMIN+0.5;
movhcurabs(&hx,&hy);
delhcur();

bx1=hx-0.5; by1=hy-0.5;
bx2=hx+0.5; by2=hy+0.5;
rbox(&bx1,&by1,&bx2,&by2);

getchr(&ch);
if(ch==0x00)
     getchr(&sch);

while(!(ch==0x1B))                    /*while not escape key*/
{
     inqhcur(&hx,&hy,&hcolor);        /*get postition of x-hair
                                        cursor*/

     if((sch==72)||(sch==75)||(sch==77)||(sch==80))
     {
          top_edge = (WYMAX+2.0)-hheight-0.05;
          bottom_edge = WYMIN+hheight;
          left_edge = WXMIN+hwidth;
          right_edge = WXMAX-hwidth-0.05;

          switch (sch)
          {
               case 72:  hy=hy+(2*hheight);  /*cursor up
                                            height x_hair*/
                         if(hy>top_edge)     /*if at top
                                               edge*/
                              hy=top_edge;   /*stay there*/
                         row[0]=row[0]-1;
                         if(row[0]<='1')
                              row[0]='1';
                         break;
               case 75:  hx=hx-(2*hwidth);   /*cursor left
                                            width x-hair*/
                         if(hx<left_edge)    /*if at left
                                               edge*/
                              hx=left_edge;  /*stay there*/
                         col[0]=col[0]-1;
                         if(col[0]<='A')
                              col[0]='A';
                         break;

               case 77:  hx=hx+(2*hwidth);   /*cursor right
                                            width x-hair*/
                         if(hx>right_edge)   /*if at right
                                               edge*/
                              hx=right_edge; /*stay there*/
                         col[0]=col[0]+1;
                         if(col[0]>='H')
```

```
                              col[0]='H';
                         break;
                    case 80:  hy=hy-(2*hheight);   /*cursor down
                                                  height x-hair*/
                              if(hy<bottom_edge)   /*if at bottom
                                                    edge*/
                                   hy=bottom_edge; /*stay there*/

                              row[0]=row[0]+1;
                              if(row[0]>='7')
                                   row[0]='7';
                              break;
                    default:  break;

               }  /*endswitch*/

          }  /*end if sch*/

          bigport();

          color=BLACK;
          setcolor(&color);
          x1=XLLROW+0.6; y1=YLLROW+0.05;
          x2=XURROW-0.05; y2=YURROW-0.05;
          bar(&x1,&y1,&x2,&y2);
          x1=XLLCOL+0.6; y1=YLLCOL+0.05;
          x2=XURCOL-0.05; y2=YURCOL-0.05;
          bar(&x1,&y1,&x2,&y2);
          color=WHITE;
          setcolor(&color);
          tx=XLLROW+0.8; ty=YLLROW+0.1;
          movtcurabs(&tx, &ty);
          text(row);
          tx=XLLCOL+0.8; ty=YLLCOL+0.1;
          movtcurabs(&tx, &ty);
          text(col);
          littleport();

          movhcurabs(&hx,&hy);
          bx1=hx-0.5; by1=hy-0.5;   /*draw cursor box*/
          bx2=hx+0.5; by2=hy+0.5;
          delhcur();
          rbox(&bx1,&by1,&bx2,&by2);


if(ch=='p'||ch=='c'||ch=='a'||ch=='v'||ch=='t'||ch=='f'||ch=='m')
          {
               inqhcur(&hx, &hy, &hcolor); /*get ctr coords of
                                             node*/

               for(i=0; i<=used_nodes+1; i=i+1)
               {
```

```
                if(i>MAXNODES)
                {
                     write_error(2);
                     break;
                }
                else if(cell[i].used != TRUE)
                {
                   active_node=i;
                   cell[i].used=TRUE;          /*mark cell
                                                 used*/
                   cell[used_nodes].nodex=hx; /*update
                                                 structure*/
                   cell[used_nodes].nodey=hy;
                   used_nodes=used_nodes+1;
                   break;
                } /*end if else*/
           } /*end for*/

      } /*end if p etc.*/

      switch(ch)
      {
          case 'p':   iconcolor=GREEN;
                      ix=hx-0.5; iy=hy-0.2;
                      pe_icon(ix, iy, iconcolor);
                      get_name();
                      element[active_node].etype="pe";
                      cell[active_node].etype="pe";
                      movhcurabs(&hx,&hy);
                      bx1=hx-0.5; by1=hy-0.5; /*draw cursor
                                                box*/
                      bx2=hx+0.5; by2=hy+0.5;
                      delhcur();
                      rbox(&bx1,&by1,&bx2,&by2);
                      break;
          case 'c':   iconcolor=BROWN;
                      ix=hx-0.5; iy=hy-0.2;
                      ce_icon(ix, iy, iconcolor);
                      get_name();
                      element[active_node].etype="ce";
                      cell[active_node].etype="ce";
                      movhcurabs(&hx,&hy);

                      bx1=hx-0.5; by1=hy-0.5;   /*draw cursor
                                                  box*/
                      bx2=hx+0.5; by2=hy+0.5;
                      delhcur();
                      rbox(&bx1,&by1,&bx2,&by2);
                      break;
          case 'a':   iconcolor=CYAN;
                      ix=hx-0.5; iy=hy-0.3;
                      a_icon(ix, iy, iconcolor);
```

```
                get_name();
                element[active_node].etype="a";
                cell[active_node].etype="a";
                movhcurabs(&hx,&hy);
                bx1=hx-0.5; by1=hy-0.5;    /*draw cursor
                                             box*/
                bx2=hx+0.5; by2=hy+0.5;
                delhcur();
                rbox(&bx1,&by1,&bx2,&by2);
                break;
        case 'v':   iconcolor=BLUE;
                ix=hx-0.5; iy=hy-0.2;
                va_icon(ix, iy, iconcolor);
                get_name();
                element[active_node].etype="va";
                cell[active_node].etype="va";
                movhcurabs(&hx,&hy);
                bx1=hx-0.5; by1=hy-0.5;    /*draw cursor
                                             box*/
                bx2=hx+0.5; by2=hy+0.5;
                delhcur();
                rbox(&bx1,&by1,&bx2,&by2);
                break;
        case 'f':   iconcolor=YELLOW;
                ix=hx-0.5; iy=hy-0.2;
                f_icon(ix, iy, iconcolor);
                get_name();
                element[active_node].etype="f";
                cell[active_node].etype="f";
                movhcurabs(&hx,&hy);
                bx1=hx-0.5; by1=hy-0.5;    /*draw cursor
                                             box*/
                bx2=hx+0.5; by2=hy+0.5;
                delhcur();
                rbox(&bx1,&by1,&bx2,&by2);
                break;
        case 't':   iconcolor=RED;
                ix=hx-0.5; iy=hy-0.2;
                t_icon(ix, iy, iconcolor);
                get_name();
                element[active_node].etype="t";
                cell[active_node].etype="t";
                movhcurabs(&hx,&hy);
                bx1=hx-0.5; by1=hy-0.5;    /*draw cursor
                                             box*/
                bx2=hx+0.5; by2=hy+0.5;
                delhcur();
                rbox(&bx1,&by1,&bx2,&by2);
                break;
        case 'm':   iconcolor=MAGENTA;
                ix=hx-1.0; iy=hy-0.2;
                m_icon(ix, iy, iconcolor);
```

```
                        get_name();
                        element[active_node].etype="m";
                        cell[active_node].etype="m";
                        movhcurabs(&hx,&hy);
                        bx1=hx-0.5; by1=hy-0.5;    /*draw cursor
                                                     box*/
                        bx2=hx+0.5; by2=hy+0.5;
                        delhcur();
                        rbox(&bx1,&by1,&bx2,&by2);
                        break;
                default:    break;
            }  /*end switch */

            if(ch=='e')
                 break;

            iconcolor=WHITE;
            setcolor(&iconcolor);

            sch=0x00;

            getchr(&ch);
            if(ch==0x00)
                 getchr(&sch);


    }  /*end while*/

    delbox();                        /*delete current rubber band
                                       box*/

    bigport();

    clear_dialog_box();

    if(ch=='e')
         add_edge();


}   /*end add_node*/


add_edge()
{
     float hheight, hwidth;              /*size of x-hair cursor*/
     float x1, y1, x2, y2;               /*graphic coordinates*/
     float cx, cy;                       /*x-hair coordinates*/
     float tx, ty;                       /*text cursor
      coordinates*/
     float bx1, by1, bx2, by2;           /*cell box coordinates*/
     float top_edge,bottom_edge;
     float left_edge,right_edge;
```

```
float lx, ly;                          /*coordinates of line's
                                         fixed end*/
int i;                                 /*loop variable*/
int turn;                              /*number of turns in an
                                         edge*/
int color;                             /*drawing color*/
int gcolor;                            /*color of graphics
                                         cursor*/
int hcolor;                            /*color of x-hair cursor*/
int line;                              /*toggle to draw/not draw
                                         line*/
char ch, sch;                          /*keyboard characters*/
char row[2], col[2];                   /*position of box cursor*/

static char message1[]=
"Move cursor to calling node and press return.";
static char message2[]=
"Or press n to add node or <esc> to return.";
static char message3[]=
"Move cursor and press t to turn or e to end edge.";
static char message4[]="  ";

write_dialog(&message1,&message2);

turn=0;

row[0]='7'; col[0]='A';
row[1]='\0';col[1]='\0';
tx=XLLROW+0.8; ty=YLLROW+0.1;
movtcurabs(&tx, &ty);
text(row);
tx=XLLCOL+0.8; ty=YLLCOL+0.1;
movtcurabs(&tx, &ty);
text(col);

ch='-'; line=FALSE;    /*init to move line, don't draw it*/

littleport();

hheight=0.5; hwidth=0.5;
hcolor=BLACK;
inithcur(&hheight,&hwidth,&hcolor);

cx=WXMIN+0.5; cy=WYMIN+0.5;  /*start cursor in lower left*/
movhcurabs(&cx,&cy);         /*corner of drawing area*/
delhcur();
bx1=cx-0.5; by1=cy-0.5;
bx2=cx+0.5; by2=cy+0.5;
rbox(&bx1,&by1,&bx2,&by2);

getchr(&ch);
if(ch==0x00)
```

```
     getchr(&sch);

while(!(ch==0x1B))                /*while not escape key*/
{
     inqhcur(&cx,&cy,&hcolor);    /*get position of x-hair
                                    cursor*/

     if((sch==72)||(sch==75)||(sch==77)||(sch==80))
     {
          top_edge = (WYMAX+2.0)-hheight-0.05;
          bottom_edge = WYMIN+hheight;
          left_edge = WXMIN+hwidth;
          right_edge = WXMAX-hwidth-0.05;

          switch (sch)
          {
               case 72:  cy=cy+(2*hheight);  /*cursor up
                                              height x_hair*/
                         if(cy>top_edge)     /*if at top
                                               edge*/
                              cy=top_edge;   /*stay there*/
                         row[0]=row[0]-1;
                         if(row[0]<='1')
                              row[0]='1';
                         break;
               case 75:  cx=cx-(2*hwidth);   /*cursor left
                                              width x-hair*/
                         if(cx<left_edge)    /*if at left
                                               edge*/
                              cx=left_edge;  /*stay there*/
                         col[0]=col[0]-1;
                         if(col[0]<='A')
                              col[0]='A';
                         break;
               case 77:  cx=cx+(2*hwidth);   /*cursor right
                                              width x-hair*/
                         if(cx>right_edge)   /*if at right
                                               edge*/
                              cx=right_edge; /*stay there*/
                         col[0]=col[0]+1;
                         if(col[0]>='H')
                              col[0]='H';
                         break;
               case 80:  cy=cy-(2*hheight);   /*cursor down
                                              height x-hair*/
                         if(cy<bottom_edge)   /*if at bottom
                                                edge*/
                              cy=bottom_edge; /*stay there*/

                         row[0]=row[0]+1;
                         if(row[0]>='7')
                              row[0]='7';
```

```
                    break;
          default:  break;

     }  /*endswitch*/

}  /*end if sch*/

bigport();
color=BLACK;
setcolor(&color);
x1=XLLROW+0.6; y1=YLLROW+0.05;
x2=XURROW-0.05; y2=YURROW-0.05;
bar(&x1,&y1,&x2,&y2);
x1=XLLCOL+0.6; y1=YLLCOL+0.05;
x2=XURCOL-0.05; y2=YURCOL-0.05;
bar(&x1,&y1,&x2,&y2);
color=WHITE;
setcolor(&color);
tx=XLLROW+0.8; ty=YLLROW+0.1;
movtcurabs(&tx, &ty);
text(row);
tx=XLLCOL+0.8; ty=YLLCOL+0.1;
movtcurabs(&tx, &ty);
text(col);
littleport();

movhcurabs(&cx,&cy);
bx1=cx-0.5; by1=cy-0.5;   /*draw cursor box*/
bx2=cx+0.5; by2=cy+0.5;
delhcur();
rbox(&bx1,&by1,&bx2,&by2);

if(ch==0x0D)
{
     bigport();            /*change viewports and*/
     color=BLACK;          /*write prompts*/
     setcolor(&color);
     write_dialog(&message3,&message4);
     littleport();         /*change viewports and*/
     movhcurabs(&cx,&cy);  /*reposition cursor*/
     delhcur();

     if(line==FALSE)
     {
          inqhcur(&cx,&cy);
          for(i=0; i<=used_nodes; i=i+1)
          {
              if(i>MAXNODES)
              {
                   write_error(2);
                   break;
              }
```

```
          else
          if((cell[i].nodex==cx)&&(cell[i].nodey==cy))
          {
                                active_node=i;
          cell[active_node].out=cell[active_node].out+1;

                                if(cell[active_node].out>MAXEDGES)
                                {
                                     write_error(2);
                                     break;
                                 } /*end if cell.out>MAXEDGES*/

                                 color=BLACK;
                                 arrow(color); /*draw arrow head*/
                                 inqgcur(&lx,&ly,&gcolor);
                                 movabs(&lx,&ly);
                                 line=TRUE;    /*fixed end of line*/

cell[active_node].edgex[cell[active_node].out][BEGIN]=lx;

cell[active_node].edgey[cell[active_node].out][BEGIN]=ly;
                                 inqhcur(&cx,&cy,&hcolor);
                                 bx1=cx-0.5; by1=cy-0.5;
                                 bx2=cx+0.5; by2=cy+0.5;
                                 rbox(&bx1,&by1,&bx2,&by2);
                                             /*delete box cursor*/
                                     movhcurabs(&cx,&cy);
                                             /*draw h-cursor*/
                                 element[active_node].rtype="calls";

element[active_node].elname[cell[active_node].out]=
    cell[active_node].ename;

element[active_node].eltype[cell[active_node].out]=
    cell[active_node].etype;
                                 break;
                            } /*end if else*/
                       } /*end for i*/

                } /*end if line is false*/

                while(line==TRUE)
                {
                      sch=0x00;
                      getchr(&ch);
                      if(ch==0x00)
                           getchr(&sch);


if((sch==72)||(sch==75)||(sch==77)||(sch==80))
                     {
                          top_edge = (WYMAX+2.0)-hheight-0.05;
```

```
                    bottom_edge = WYMIN+hheight;
                    left_edge = WXMIN+hwidth;
                    right_edge = WXMAX-hwidth-0.05;

                    switch (sch)
                    {
                         case 72:  cy=cy+(2*hheight);
                                   if(cy>top_edge)
                                   cy=top_edge;
                                   break;
                         case 75:  cx=cx-(2*hwidth);
                                   if(cx<left_edge)
                                   cx=left_edge;
                                   break;
                         case 77:  cx=cx+(2*hwidth);
                                   if(cx>right_edge)
                                   cx=right_edge;
                                   break;
                         case 80:  cy=cy-(2*hheight);
                                   if(cy<bottom_edge)
                                   cy=bottom_edge;
                                   break;
                         default:  line=FALSE;
                                   break;

                    }  /*endswitch*/

               }  /*end if sch*/

               if(ch=='t')
               {
                    turn=turn+1;
                    if(turn>3)
                    {
                         write_error(2);
                         break;
                    }
                    color=BLACK;
                    setcolor(&color);
                    delhcur();
                    lx=cx; ly=cy;  /*draw line from fixed
                                     end*/
                    lnabs(&lx,&ly);       /*x-hair cursor
                                            position*/
                    line=TRUE;

cell[active_node].edgex[cell[active_node].out][turn]=lx;

cell[active_node].edgey[cell[active_node].out][turn]=ly;

               }  /*end if ch==t*/
```

```
                    if(ch=='e')
                    {
                         color=BLACK;
                         setcolor(&color);
                         delhcur();
                         lx=cx; ly=cy+0.55;     /*draw line from
                                                  fixed end*/
                         lnabs(&lx,&ly);        /*x-hair cursor
                                                  position*/
                         line=FALSE;

cell[active_node].edgex[cell[active_node].out][END]=lx;

cell[active_node].edgey[cell[active_node].out][END]=ly;

                         for(i=1;i<=used_nodes;i=i+1)
                         {

if((cell[i].nodex==cx)&&(cell[i].nodey==cy));
                                   {

cell[i].cnode[cell[active_node].out]=active_node;

cell[i].cedge[cell[active_node].out]=cell[active_node].out;

element[i].e2name[cell[i].in]=cell[i].ename;

element[i].e2type[cell[i].in]=cell[i].etype;
                                   }
                         }  /*end for*/
                    } /*end if ch=e*/

                     movhcurabs(&cx,&cy);

               } /*end while line is true*/
          }  /*end if return*/

          turn=0;              /*reset turns in edges to zero*/

          sch=0x00;

          getchr(&ch);
          if(ch==0x00)
               getchr(&sch);

          if(ch=='n')
               break;
     }  /*end while*/

     bigport();

     if(ch=='n')
```

```
          add_node();

}  /*end add edge*/


change_name()
{
     float x1, y1, x2, y2;                /*graphic coordinates*/

     float bx1, by1, bx2, by2;           /*cell box coordinates*/

     float cx, cy;                     /*x-hair cursor coordinates*/

     float hheight, hwidth;          /*x-hair cursor height/width*/

     float top_edge,bottom_edge;
     float left_edge,right_edge;
     int i;                                    /*loop variable*/
     int hcolor;                            /*x-hair cursor color*/
     int color;                           /*current drawing color*/
     char ch, sch;                          /*keyboard characters*/
     static char message1[]=
     "Move cursor to desired node and press return.";
     static char message2[]=" ";

     write_dialog(&message1,&message2);

     littleport();                          /*reset viewport area*/

     hheight=0.5; hwidth=0.5;
     hcolor=BLACK;
     inithcur(&hheight,&hwidth,&hcolor);

     cx=WXMIN+0.5; cy=WYMIN+0.5;
     movhcurabs(&cx,&cy);
     delhcur();

     bx1=cx-0.5; by1=cy-0.5;
     bx2=cx+0.5; by2=cy+0.5;
     rbox(&bx1,&by1,&bx2,&by2);

     getchr(&ch);
     if(ch==0x00)
          getchr(&sch);

     while(!(ch==0x1B))                  /*while not escape key*/
     {
          inqhcur(&cx,&cy,&hcolor);      /*get postition of x-hair
                                           cursor*/

          if((sch==72)||(sch==75)||(sch==77)||(sch==80))
          {
```

```
        top_edge = (WYMAX+2.0)-hheight;
        bottom_edge = WYMIN+hheight;
        left_edge = WXMIN+hwidth;
        right_edge = WXMAX-hwidth-0.05;

        switch (sch)
        {
             case 72:  cy=cy+(2*hheight);  /*cursor up
                                          height x_hair*/
                       if(cy>top_edge)     /*if at top
                                             edge*/
                            cy=top_edge;   /*stay there*/
                       break;
             case 75:  cx=cx-(2*hwidth);   /*cursor left
                                          width x-hair*/
                       if(cx<left_edge)    /*if at left
                                             edge*/
                            cx=left_edge;  /*stay there*/
                       break;
             case 77:  cx=cx+(2*hwidth);   /*cursor right
                                          width x-hair*/
                       if(cx>right_edge)   /*if at right
                                             edge*/
                            cx=right_edge; /*stay there*/
                       break;
             case 80:  cy=cy-(2*hheight);   /*cursor down
                                          height x-hair*/
                       if(cy<bottom_edge)   /*if at bottom
                                              edge*/
                            cy=bottom_edge; /*stay there*/

                       break;
             default:  break;

        }  /*endswitch*/

   }  /*end if sch*/

   movhcurabs(&cx,&cy);
   bx1=cx-0.5; by1=cy-0.5;   /*draw cursor box*/
   bx2=cx+0.5; by2=cy+0.5;
   delhcur();
   rbox(&bx1,&by1,&bx2,&by2);

   if(ch==0x0D)              /*if return*/
   {
        inqhcur(&cx,&cy,&hcolor);

        for(i=0; i<=used_nodes; i=i+1)
        {
             if((cell[i].nodex==cx)&&(cell[i].nodey==cy))
                  active_node=i;
```

```
                }  /*end for*/

                color=WHITE;
                setcolor(&color);
                x1=cx-0.45; y1=cy-0.45;   /*delete old name*/
                x2=cx+0.45; y2=cy-0.25;
                bar(&x1,&y1,&x2,&y2);

                get_name();                /*get new name*/

           }  /*end if ch=return*/


           sch=0x00;

           getchr(&ch);
           if(ch==0x00)
                getchr(&sch);

     }  /*end while*/

     delbox();          /*delete current rubber band box*/

     bigport();

}    /*end change_name*/


change_type()
{
      float x1, y1, x2, y2;                /*graphic coordinates*/
      float ix, iy;                       /*icon area coordinates*/
      float bx1, by1, bx2, by2;           /*cell box coordinates*/
      float cx, cy;                    /*x-hair cursor coordinates*/
      float hheight, hwidth;          /*x-hair cursor height/width*/
      float top_edge,bottom_edge;
      float left_edge,right_edge;
      int hcolor;                          /*x-hair cursor color*/
      int color;                          /*current drawing color*/
      int iconcolor;                       /*current icon color*/
      char ch, sch;                        /*keyboard characters*/

      littleport();                        /*reset viewport area*/

      hheight=0.5; hwidth=0.5;
      hcolor=BLACK;
      inithcur(&hheight,&hwidth,&hcolor);

      cx=WXMIN+0.5; cy=WYMIN+0.5;
      movhcurabs(&cx,&cy);
      delhcur();
```

```
bx1=cx-0.5; by1=cy-0.5;
bx2=cx+0.5; by2=cy+0.5;
rbox(&bx1,&by1,&bx2,&by2);

getchr(&ch);
if(ch==0x00)
     getchr(&sch);

while(!(ch==0x1B))                    /*while not escape key*/
{
     inqhcur(&cx,&cy,&hcolor);        /*get postition of x-hair
                                        cursor*/

     if((sch==72)||(sch==75)||(sch==77)||(sch==80))
     {
          top_edge = (WYMAX+2.0)-hheight;
          bottom_edge = WYMIN+hheight;
          left_edge = WXMIN+hwidth;
          right_edge = WXMAX-hwidth-0.05;

          switch (sch)
          {
               case 72:  cy=cy+(2*hheight);  /*cursor up
                                             height x_hair*/
                         if(cy>top_edge)     /*if at top
                                               edge*/
                              cy=top_edge;   /*stay there*/
                         break;
               case 75:  cx=cx-(2*hwidth);   /*cursor left
                                             width x-hair*/
                         if(cx<left_edge)    /*if at left
                                               edge*/
                              cx=left_edge;  /*stay there*/
                         break;
               case 77:  cx=cx+(2*hwidth);   /*cursor right
                                             width x-hair*/
                         if(cx>right_edge)   /*if at right
                                               edge*/
                              cx=right_edge; /*stay there*/
                         break;

               case 80:  cy=cy-(2*hheight);   /*cursor down
                                             height x-hair*/
                         if(cy<bottom_edge)   /*if at bottom
                                                edge*/
                              cy=bottom_edge; /*stay there*/

                         break;
               default:  break;

          }  /*endswitch*/
     }  /*end if sch*/
```

```
movhcurabs(&cx,&cy);
bx1=cx-0.5; by1=cy-0.5;   /*draw cursor box*/
bx2=cx+0.5; by2=cy+0.5;
delhcur();
rbox(&bx1,&by1,&bx2,&by2);

if(ch=='t')
{
     getchr(&ch);

     inqhcur(&cx,&cy);

     color=WHITE;
     setcolor(&color);
     x1=cx-0.45; y1=cy-0.2;   /*draw cursor box*/
     x2=cx+0.45; y2=cy+0.45;
     bar(&x1,&y1,&x2,&y2);

     switch(ch)
     {
          case 'p':   iconcolor=GREEN;
                      ix=cx-0.5; iy=cy-0.2;
                      pe_icon(ix, iy, iconcolor);
                      break;
          case 'c':   iconcolor=BROWN;
                      ix=cx-0.5; iy=cy-0.2;
                      ce_icon(ix, iy, iconcolor);
                      break;
          case 'a':   iconcolor=CYAN;
                      ix=cx-0.5; iy=cy-0.2;
                      a_icon(ix, iy, iconcolor);
                      break;
          case 'v':   iconcolor=BLUE;
                      ix=cx-0.5; iy=cy-0.2;
                      va_icon(ix, iy, iconcolor);
                      break;
          case 'f':   iconcolor=YELLOW;
                      ix=cx-0.5; iy=cy-0.2;
                      f_icon(ix, iy, iconcolor);
                      break;
          case 't':   iconcolor=RED;
                      ix=cx-0.5; iy=cy-0.2;
                      t_icon(ix, iy, iconcolor);
                      break;
          case 'm':   iconcolor=MAGENTA;
                      ix=cx-0.5; iy=cy-0.2;
                      m_icon(ix, iy, iconcolor);
                      break;
          default:    break;
     }  /*end switch */

}  /*end if ch=t*/
```

```
            sch=0x00;

            getchr(&ch);
            if(ch==0x00)
                 getchr(&sch);

        }  /*end while*/

        delbox();                /*delete current rubber band box*/

        bigport();

    }   /*end change_type*/
```

```
/*************************************************************
*  Name:     sfun2.c                                         *
*  Purpose:  contains sub-functions called by primary command *
*            functions constained in file fun.c              *
*  Author:   David D. O'Dell                                 *
*  Date:     7 January 1988                                  *
*************************************************************/

#include "intuit.h"
#include "exstruct.h"



create_module_tree()
{
     char ch, sch;
     int on;
     static char message1[]="FUNCTION NOT IMPLEMENTED.";
     static char message2[]="Press any key to continue.";

     write_dialog(&message1,&message2);

     getchr(&ch);

     if(ch==0x00)
     {
          getchr(&sch);
          on=decode_char(sch);
     } /*end if 0x00*/

     clear_dialog_box();

     return(on);

}  /*end creat_module_tree*/


edit_model_paragraphs()
{
     char ch, sch;            /*keyboard characters*/
     int record;
     int segment;
     int address;
     int on;
     int color;
     float x1,y1,x2,y2;

     buf32(&segment);
     address=segment*12;
     imsave(&address);         /*save screen to memory*/

     color=BLACK;
```

```
        setcolor(&color);
        clr();                  /*clear entire screen to BLACK*/

        color=RED;
        setcolor(&color);
        x1=0.1, y1=0.1, x2=WXMAX-0.1; y2=WYMAX-0.1;
        box(&x1,&y1,&x2,&y2);
        x1=0.1; y1=WYMAX-0.5;
        movabs(&x1,&y1);
        x2=WXMAX-0.1; y2=WYMAX-0.5;
        lnabs(&x2,&y2);
        x1=0.1; y1=0.5;
        movabs(&x1,&y1);
        x2=WXMAX-0.1; y2=0.5;
        lnabs(&x2,&y2);

        record=active_node;

        curlocate(1,6);
        printf("MODEL  NAME:  %-12s",model_name);
        curlocate(1,56);
        printf("RECORD NUMBER:  %-4d",record);
        curlocate(3,2);
        printf("Name:  %-12s",element[record].ename);
        curlocate(3,34);
        printf("Description:  %-30s",element[record].dname);
        curlocate(4,2);
        printf("Type:  %-8s",element[record].etype);
        curlocate(5,2);
        printf("Date Added:  %-7s",element[record].date_added);
        curlocate(5,34);
        printf("Last Modified:  %-7s",element[record].last_mod);
        curlocate(5,59);
        printf("Number Mods:  %-5d",element[record].nmods);
        curlocate(7,2);
        printf("Index:  %-4s",element[record].idx);
        curlocate(7,34);
        printf("Generic Range:  %-20s",element[record].grange);
        curlocate(8,2);
        printf("Index Statement:  %-58s",element[record].idx_stmt);
        curlocate(11,2);
        printf("Generic Rule:  %-61s",element[record].grule);
        curlocate(14,2);
        printf("Comments:  %-65s",element[record].comments);
        curlocate(17,2);
        printf("Relationship Type: %-8s",element[record].rtype);
        curlocate(17,31);
        printf("Edge 1:   %-12s  called by
%-12s",element[record].e2name[1],
                                element[record].elname[1]);
        curlocate(18,31);
        printf("Edge 2:   %-12s  called by
```

```
                        %-12s",element[record].e2name[2],
                        element[record].e1name[2]);
curlocate(19,31);
printf("Edge 3:   %-12s  called by
                        %-12s",element[record].e2name[3],
                        element[record].e1name[3]);
curlocate(20,2);
printf("Relative Position: %-8d",element[record].rel_pos);
curlocate(20,31);
printf("Edge 4:   %-12s  called by
                        %-12s",element[record].e2name[4],
                        element[record].e1name[4]);
curlocate(21,31);
printf("Edge 5:   %-12s  called by
                        %-12s",element[record].e2name[5],
                        element[record].e1name[5]);

curlocate(23,8);
printf("[+] Next  [-] Previous  [e] Edit  [f] find");
printf("  [s] Save  [r]  Return");


getchr(&ch);
if(ch==0x00)
{
    getchr(&sch);
    on=decode_char(sch);
}  /*end if 0x00*/

while(!(ch==0x1B))              /*while not escape*/
{
     if(ch=='+')
     {
          record=record+1;
          if(record==MAXNODES)
               record=0;
          curlocate(1,70);
          printf("  %-4d",record);
          curlocate(3,9);
          printf("%-12s",element[record].ename);
          curlocate(3,48);
          printf("%-30s",element[record].dname);
          curlocate(4,9);
          printf("%-8s",element[record].etype);
          curlocate(5,15);
          printf("%-7s",element[record].date_added);
          curlocate(5,50);
          printf("%-7s",element[record].last_mod);
          curlocate(5,73);
          printf("%-5d",element[record].nmods);
          curlocate(7,10);
          printf("%-4s",element[record].idx);
```

```
            curlocate(7,50);
            printf("%-20s",element[record].grange);
            curlocate(8,20);
            printf("%-58s",element[record].idx_stmt);
            curlocate(11,17);
            printf("%-61s",element[record].grule);
            curlocate(14,13);
            printf("%-65s",element[record].comments);
            curlocate(17,21);
            printf("%-8s",element[record].rtype);
            curlocate(17,40);
            printf(" %-12s  called by
                        %-12s",element[record].e2name[1],
                        element[record].elname[1]);
            curlocate(18,40);
            printf(" %-12s  called by
                        %-12s",element[record].e2name[2],
                        element[record].elname[2]);
            curlocate(19,40);
            printf(" %-12s  called by
                        %-12s",element[record].e2name[3],
                        element[record].elname[3]);
            curlocate(20,21);
            printf("%-8d",element[record].rel_pos);
            curlocate(20,40);
            printf(" %-12s  called by
                        %-12s",element[record].e2name[4],
                        element[record].elname[4]);
            curlocate(21,40);
            printf(" %-12s  called by
                        %-12s",element[record].e2name[5],
                        element[record].elname[5]);

      }
      else if(ch=='-')
      {
            record=record-1;
            if(record<0)
                  record=MAXNODES-1;
            curlocate(1,70);
            printf("  %-4d",record);
            curlocate(3,9);
            printf("%-12s",element[record].ename);
            curlocate(3,48);
            printf("%-30s",element[record].dname);
            curlocate(4,9);
            printf("%-8s",element[record].etype);
            curlocate(5,15);
            printf("%-7s",element[record].date_added);
            curlocate(5,50);
            printf("%-7s",element[record].last_mod);
            curlocate(5,73);
```

```
        printf("%-5d",element[record].nmods);
        curlocate(7,10);
        printf("%-4s",element[record].idx);
        curlocate(7,50);
        printf("%-20s",element[record].grange);
        curlocate(8,20);
        printf("%-58s",element[record].idx_stmt);
        curlocate(11,17);
        printf("%-61s",element[record].grule);
        curlocate(14,13);
        printf("%-65s",element[record].comments);
        curlocate(17,21);
        printf("%-8s",element[record].rtype);
        curlocate(17,40);
        printf(" %-12s  called by
                %-12s",element[record].e2name[1],
                element[record].elname[1]);
        curlocate(18,40);
        printf(" %-12s  called by
                %-12s",element[record].e2name[2],
                element[record].elname[2]);
        curlocate(19,40);
        printf(" %-12s  called by
                %-12s",element[record].e2name[3],
                element[record].elname[3]);
        curlocate(20,21);
        printf("%-8d",element[record].rel_pos);
        curlocate(20,40);
        printf(" %-12s  called by
                %-12s",element[record].e2name[4],
                element[record].elname[4]);
        curlocate(21,40);
        printf(" %-12s  called by
                %-12s",element[record].e2name[5],
                element[record].elname[5]);

    }
    else if(ch=='f')
    {
        curlocate(23,8);
        printf("Find not implemented--press any key to
                continue.");
        getchr(&ch);
        curlocate(23,8);
        printf("[+] Next  [-] Previous  [e] Edit  [f]
                find");
        printf("  [s] Save  [r]  Return");
    }
    else if(ch=='s')
    {
        curlocate(23,8);
```

```
        printf("Save not implemented--press any key to
                continue.");
        getchr(&ch);
        curlocate(23,8);
        printf("[+] Next  [-] Previous  [e] Edit  [f]
                find");
        printf("  [s] Save  [r]  Return");
}
else if(ch=='e')
{
        curlocate(23,8);
        printf("<TAB> to move between fields, <ESC> to
                exit editing.");

        getchr(&ch);
             if(ch=0x00)
                  getchr(&sch);
        while(!(ch==0x1B))          /*while not escape*/
        {
             curlocate(3,33);
             printf(">");
             getchr(&ch);
                 if(ch=0x00)
                      getchr(&sch);
             if(ch==0x1B)
                  break;
             else if(!(sch==0x0F))      /*if not tab*/
             {
                  curlocate(3.48);      /*description*/
                  scanf("%s",element[record].dname);
             }  /*end if else*/

             curlocate(5,1);
             printf(">");
             getchr(&ch);
                 if(ch=0x00)
                      getchr(&sch);
             if(ch==0x1B)
                  break;
             else if(!(sch==0x0F))      /*if not tab*/
             {
                  curlocate(5,15);      /*date added*/
                  scanf("%s",element[record].date_added);
             }  /*end if else*/

             curlocate(5,33);
             printf(">");
             getchr(&ch);
                 if(ch=0x00)
                      getchr(&sch);
             if(ch==0x1B)
                  break;
```

```
else if(!(sch==0x0F))      /*if not tab*/
{
     curlocate(5,50);      /*last mod*/
     scanf("%s",element[record].last_mod);
}  /*end if else*/

curlocate(5,58);
printf(">");
getchr(&ch);
    if(ch=0x00)
         getchr(&sch);
if(ch==0x1B)
     break;
else if(!(sch==0x0F))      /*if not tab*/
{
     curlocate(5,73);      /*num mods*/
     scanf("%s",element[record].nmods);
}  /*end if else*/

curlocate(7,1);
printf(">");
getchr(&ch);
    if(ch=0x00)
         getchr(&sch);
if(ch==0x1B)
     break;
else if(!(sch==0x0F))      /*if not tab*/
{
     curlocate(7,9);      /*index*/
     scanf("%s",element[record].idx);
}  /*end if else*/

curlocate(7,33);
printf(">");
getchr(&ch);
    if(ch=0x00)
         getchr(&sch);
if(ch==0x1B)
     break;
else if(!(sch==0x0F))      /*if not tab*/
{
     curlocate(7,50);      /*generic range*/
     scanf("%s",element[record].grange);
}  /*end if else*/

curlocate(8,1);
printf(">");
getchr(&ch);
    if(ch=0x00)
         getchr(&sch);
if(ch==0x1B)
     break;
```

```
                else if(!(sch==0x0F))      /*if not tab*/
                {
                     curlocate(8,20);      /*index statement*/

                     scanf("%s",element[record].idx_stmt);
                }  /*end if else*/

                curlocate(11,1);
                printf(">");
                getchr(&ch);
                    if(ch=0x00)
                         getchr(&sch);
                if(ch==0x1B)
                     break;
                else if(!(sch==0x0F))      /*if not tab*/
                {
                     curlocate(11,17);      /*generic rule*/
                     scanf("%s",element[record].grule);
                }  /*end if else*/

                curlocate(14,1);
                printf(">");
                getchr(&ch);
                    if(ch=0x00)
                         getchr(&sch);
                if(ch==0x1B)
                     break;
                else if(!(sch==0x0F))      /*if not tab*/
                {
                     curlocate(14,13);      /*comments*/
                     scanf("%s",element[record].comments);
                }  /*end if else*/

          }  /*end while not escape*/

          curlocate(23,8);
          printf("[+] Next  [-] Previous  [e] Edit  [f]
                  find");
          printf("  [s] Save  [r]  Return");

     } /*end if ch=e*/
     else if(ch=='r')
     {
          break;
     } /*end if else*/

     getchr(&ch);
     if(ch==0x00)
     {
          getchr(&sch);
          on=decode_char(sch);
     }  /*end if 0x00*/
```

```
     } /*end while not escape*/


     imrest(&address);     /*restore screen from memory*/

     on=1;
     return(on);

}   /*end edit model paragraphs*/


add_module()
{
     char ch;
     static char message1[]="FUNCTION NOT IMPLEMENTED.";
     static char message2[]="Press any key to continue.";

     write_dialog(&message1,&message2);

     getchr(&ch);

     clear_dialog_box();

}


delete_node()
{
     float x1, y1, x2, y2;               /*graphic coordinates*/
     float bx1, by1, bx2, by2;           /*cell box coordinates*/

     float cx, cy;                  /*x-hair cursor coordinates*/
     float hheight, hwidth;        /*x-hair cursor height/width*/
     float top_edge,bottom_edge;
     float left_edge,right_edge;
     int hcolor;                         /*x-hair cursor color*/
     int color;                         /*current drawing color*/
     char ch, sch;                       /*keyboard characters*/
     static char message1[]="Move cursor and press return to
                  delete node.";
     static char message2[]="Or press <esc> to return.";

     write_dialog(&message1,&message2);

     littleport();                       /*reset viewport area*/

     hheight=0.5; hwidth=0.5;
     hcolor=BLACK;
     inithcur(&hheight,&hwidth,&hcolor);

     cx=WXMIN+0.5; cy=WYMIN+0.5;
```

```
movhcurabs(&cx,&cy);
delhcur();

bx1=cx-0.5; by1=cy-0.5;
bx2=cx+0.5; by2=cy+0.5;
rbox(&bx1,&by1,&bx2,&by2);

getchr(&ch);
if(ch==0x00)
     getchr(&sch);

while(!(ch==0x1B))                    /*while not escape key*/
{
     inqhcur(&cx,&cy,&hcolor);        /*get postition of x-hair
                                        cursor*/

     if((sch==72)||(sch==75)||(sch==77)||(sch==80))
     {
          top_edge = (WYMAX+2.0)-hheight;
          bottom_edge = WYMIN+hheight;
          left_edge = WXMIN+hwidth;
          right_edge = WXMAX-hwidth-0.05;

          switch (sch)
          {
               case 72:  cy=cy+(2*hheight);  /*cursor up
                                              height x_hair*/
                         if(cy>top_edge)     /*if at top
                                               edge*/
                              cy=top_edge;   /*stay there*/
                         break;
               case 75:  cx=cx-(2*hwidth);   /*cursor left
                                              width x-hair*/
                         if(cx<left_edge)    /*if at left
                                               edge*/
                              cx=left_edge;  /*stay there*/
                         break;
               case 77:  cx=cx+(2*hwidth);   /*cursor right
                                              width x-hair*/
                             if(cx>right_edge)   /*if at
                                                  right edge*/
                              cx=right_edge; /*stay there*/
                         break;
               case 80:  cy=cy-(2*hheight);   /*cursor down
                                              height x-hair*/
                         if(cy<bottom_edge)   /*if at bottom
                                                edge*/
                              cy=bottom_edge; /*stay there*/

                         break;
               default:  break;
```

```
                }  /*endswitch*/

            }  /*end if sch*/

            movhcurabs(&cx,&cy);
            bx1=cx-0.5; by1=cy-0.5;    /*draw cursor box*/
            bx2=cx+0.5; by2=cy+0.5;
            delhcur();
            rbox(&bx1,&by1,&bx2,&by2);

            if(ch==0x0D)               /*if return*/
            {
                 inqhcur(&cx,&cy);

                 color=WHITE;
                 setcolor(&color);
                 x1=cx-0.45; y1=cy-0.45;    /*draw cursor box*/
                 x2=cx+0.45; y2=cy+0.45;
                 bar(&x1,&y1,&x2,&y2);
            }  /*end if ch=d*/

            sch=0x00;

            getchr(&ch);
            if(ch==0x00)
                 getchr(&sch);

    }  /*end while*/

    delbox();              /*delete current rubber band box*/

    bigport();

    clear_dialog_box();

}   /*end delete_node*/



delete_edge()
{
     char ch;
     static char message1[]="FUNCTION NOT IMPLEMENTED.";
     static char message2[]="Press any key to continue.";

     write_dialog(&message1,&message2);

     getchr(&ch);

     clear_dialog_box();

}
```

```
delete_module()
{
     char ch;
     static char message1[]="FUNCTION NOT IMPLEMENTED.";
     static char message2[]="Press any key to continue.";

     write_dialog(&message1,&message2);

     getchr(&ch);

     clear_dialog_box();

}
```

APPENDIX D

INTUITION
User's Manual

TABLE OF CONTENTS **Page**

1: INTRODUCTION

Welcome to INTUITION, a graphics-based program that allows you to generate sophisticated model representations directly on the computer screen and to transform these representations to a database form for subsequent manipulation and solution. INTUITION assumes that you are familiar with the concept of Structured Modeling and makes no attempt to "teach" this concept itself. Otherwise, the functions of INTUITION are sufficiently simple to master with a few minutes of practice. The program is totally self-prompting and menu-driven. We have tried to make it work in an manner that is both obvious and intuitive (hence the name). Normally, what you expect to happen will happen; but, if something does go wrong, the program will identify the error and tell you what to do to fix it. You do not need to read the user's manual from cover to cover in order to use the program. In fact, you would find much of the reading duplicative. The user's manual is intended as a reference manual, so when you look up a particular topic, all of the information for using that function is contained within that section. You don't have to go on a scavenger hunt to find out what you need to know. So, with that in mind, let's begin....

2: SYSTEM REQUIREMENTS

HARDWARE:

An IBM PC (XT or AT) or compatible with at least one disk drive.

A monitor capable of resolving, at a minimum, 640 x 200 pixels.

SOFTWARE:

An INTUITION (V 1.1) program disk.

HALO Graphics Library (specifically, the graphics driver for your particular computer). If not previously installed, the appropriate driver must be copied to the INTUITION program disk.

ORACLE Relational Database Management System (RDBMS) for IBM PC/MS-DOS

MEMORY:

The program will run minimally in 640K of memory. For effective operation, expanded memory of 1M or more is recommended. (This does not consider the memory needed to run the ORACLE RDBMS. Consult your ORACLE User's Guide to determine the RDBMS's memory requirements.)

OPERATING SYSTEM:

IBM PC DOS or MSDOS (Release 2.0 or later).

GRAPHIC DISPLAY CARDS:

IBM Color Graphics Adapter (CGA)
Generic CGA work-alikes
IBM Enhanced Graphics Adapter (EGA)
Sigma Designs Color 400

POINTING DEVICES:

Keyboard cursor keys (default)
Microsoft or compatible mouse (optional)

3: GETTING STARTED

POWER UP:

Ensure that the appropriate graphics driver has been copied to the INTUITION program disk. (If using a mouse, ensure that the mouse driver has also been copied.)

Insert the INTUITION program disk in drive A: and type **int** to invoke the program. An initial menu screen will quickly appear. Enter the number of the graphics device that matches your equipment configuration and press return. The main program will then be loaded.

THE OPENING SCREEN:

Shortly you will be presented with an opening screen similar to Figure 1. This screen consists of five windows:

At the top of the screen is the status window. This shows the type and name of the current model.

The window on the top right contains the geometric shapes used to draw the structured models.

The window on the bottom right contains the command buttons available in the program.

The window at the bottom of the screen is the dialog area. Prompts and error messages will always appear here.

The white center area is the workspace for drawing the structured model.

THE DRAWING ELEMENTS:

Each icon on the right side of the screen represents different type of element in the structured model.

PE represents a primitive entity element and is a green square.

CE represents a compound entity element and is a brown octagon.

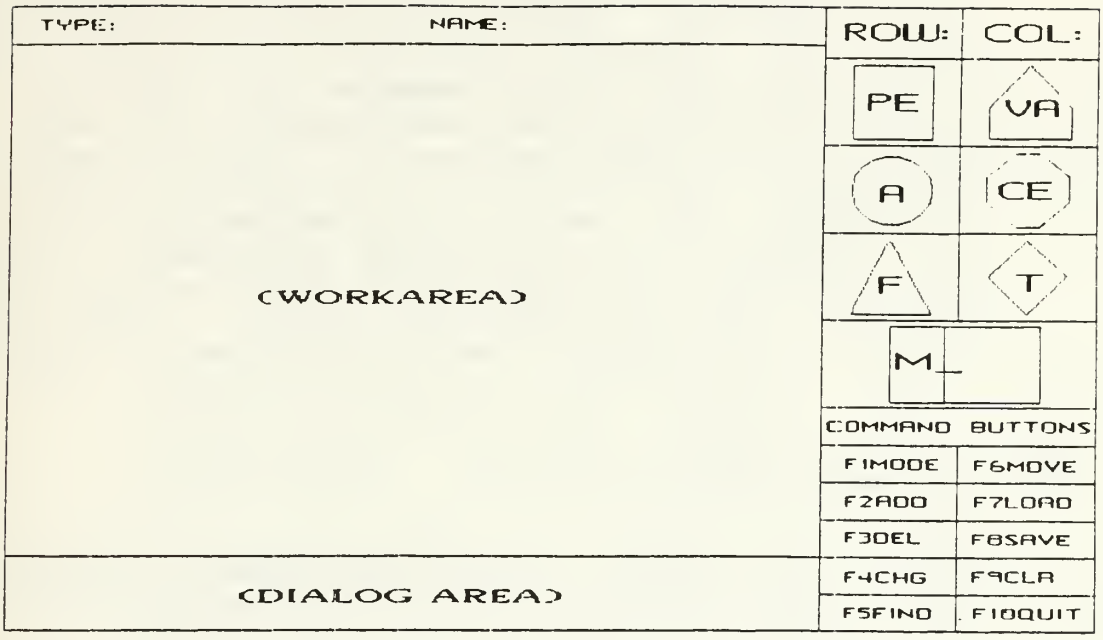

Figure 1: Main Screen

A represents an attribute element and is a cyan circle.

VA represents a variable attribute element and is a blue trapezoid.

F represents a function element and is a yellow triangle.

T represents a test element and is a red diamond.

M represents a module and is a magenta rectangle.

(In monochrome mode, all icons are white with a black border.)

THE COMMAND BUTTONS:

All program functions are initiated by choosing the appropriate command button. The selected button is highlighted. Only one command can be selected at a time. Commands are chosen by pressing the appropriate function key (e.g. f1 function key selects F1Mode), by moving the highlight with the cursor keys, or by clicking on the desired command with the left mouse button. Commands may be selected in any order. (When using the cursor keys to move to a specific command, all intervening command buttons are rapidly selected and deselected. This can be disconcerting at times. For this reason, using the function keys and/or mouse to select commands is recommended.) Some commands produce submenus, providing additional choices. Simply follow the appropriate prompts when presented with a submenu.

HOW TO USE YOUR POINTING DEVICE:

The primary purposes of a pointing device are to move the various "cursors" around the screen, to draw on the screen, and to select icons and commands from the menus.

The default pointing device is the keyboard cursor keys. To access these keys, the numeric keypad must be in "cursor mode." If your computer prints numeric digits on the screen when you press a cursor key, press the "num lock" key once to exit "numeric mode." Then pressing the cursor keys will move the cursor either horizontally or vertically.

A Microsoft compatible mouse may also be used as an optional pointing device in future versions of the program. Ensure that the mouse is connected to the computer (normally through the serial port) and that the appropriate HALO graphics mouse driver is on the program disk. To move the cursor, simply move the mouse in the direction you want the cursor to move. To select or drag an object, click (press) the left mouse button. To release or place an object, click the right mouse-button. (If you have a three button mouse, the middle button is currently unused.)

SELECTING THE PROGRAM MODE:

The program will display the F1Mode submenu:

The program will prompt you to enter a model name. The name must be 10 or less characters long. If you attempt to enter more than 10 characters, only the first 10 will be accepted and the program will automatically continue from this point. Valid names can contain any combination of letters, digits or underscores. End names of less than 10 characters by pressing the return key.

Press **1** to enter a genus graph. The program prepares the drawing area and data structures to accept a new genus graph. WARNING: SELECTING THIS OPTION DESTROYS ANY PREVIOUS GENUS GRAPH IN THE DRAWING AREA.

Press **2** to enter a modular tree. The program prepares the drawing area and data structures to accept a new genus graph. WARNING: SELECTING THIS OPTION DESTROYS ANY PREVIOUS MODULAR TREE IN THE DRAWING AREA.

Press **3** to edit module or genus paragraphs. The program will display the paragraph editing screen. You can scroll through the model paragraphs by using the + and - keys. You can move from field to field in a specific paragraph by using the left and right cursor keys. You can edit any field except genus name, module name, genus type, and calling sequence. These fields can only be modified by changing the appropriate genus graph or module tree. The cursor will not enter one of these fields on the edit screen. Press the **escape key** to exit editing mode. This will return you to the initial display screen.

ADDING ELEMENTS TO THE MODEL:

Select the F2Add command button by either pressing the f2 function key, clicking on the F2Add command with the left mouse button, or moving the highlight to F2Add with the cursor keys. The program will display the F2Add submenu:

Press **1** to add an element to the genus graph.

A box cursor will appear in the lower left corner of the drawing area. Move the cursor with the cursor keys or mouse. It will move within a grid eight blocks wide by seven blocks high.

To enter an element, move the box cursor to the desired location and press the first letter of the element type (i.e., **p**, **c**, **a**, **v**, **f**, **t** or **m**) or click on the element icon with the left mouse button. The appropriate icon will be drawn inside the box cursor.

The program will then prompt you to enter an element name. Names must be seven or less characters long. If you attempt to enter more than seven characters, only the first seven will be accepted and the program will automatically continue from this point. Valid genus names must be unique and must begin with a letter. The remainder of the name may contain any combination of letters, digits or underscores. End names of less than seven characters by pressing the return key.

ELEMENT NAMES ARE MANDATORY ENTRIES. YOU CANNOT CONTINUE UNTIL YOU ENTER A VALID ELEMENT NAME.

You can now move the box cursor to enter another element or:

press **e** to add an edge;
press **m** to add a module;
press the **escape key** to return to the F2Add submenu.

To return to adding nodes after adding edges and/or modules, simply press **n**. You can repeatedly cycle between adding nodes, edges, or modules by pressing the **n**, **e**, or **m** keys.

Press **2** to add an edge (directed arc) to the genus graph or an undirected arc to a modular tree.

A box cursor will appear in the lower left corner of the drawing area. Move the cursor with the cursor keys or mouse. It will move within a grid eight blocks wide by seven blocks high.

To draw an edge, move the box cursor to the calling element and press the **return key**. The program will automatically draw the "arrow" end of a directed arc below the element icon in a genus graph. Or it will begin an undirected arc to the left side of a modular icon in a modular tree.

The box cursor will change to a cross-hair cursor. Move the cross-hair cursor (with the cursor keys or mouse) to the called element and press **e** for end. The edge will be drawn. (Note: When using the cursor keys to draw an edge, you can only move the cursor horizontally or vertically; however, the actual edge will be a straight line directly from the beginning point to the end point. It will NOT be a series of vertical and horizontal lines following the cursor path.)

Sometimes, when an edge is drawn directly between two elements, if will cross an intervening element. To prevent this, an edge can contain up to three turns. To turn an edge, press **t** for turn. The completed edge segment will be drawn. You can then continue the edge in any direction. This allows you to draw "around corners."

You can now move the box cursor to enter another edge or:

press **n** to add a node;
press **m** to add a module;
press the **escape key** to return to the F2Add submenu.

To return to adding edges after adding nodes and/or modules, simply press **e**. You can repeatedly cycle between adding nodes, edges, or modules by pressing the **n**, **e**, or **m** keys.

Press **3** to add a module to the module tree or genus graph.

A rectangular cursor will appear in the lower left corner of the drawing area. Move the cursor with the cursor keys or mouse. It will move within a grid eight blocks wide by seven blocks high.

To enter a module, move the rectangular cursor to the desired location and press **m** or click the left mouse button. The module icon will be drawn inside the rectangular cursor.

The program will then prompt you to enter an module name. Names must be seven or less characters long. If you attempt to enter more than seven characters, only the first seven will be accepted and the program will automatically continue from this point. Valid module names must be unique and must begin with a capital M and underline character (M_). The remainder of the name may contain any combination of letters, digits or underscores. End names of less than seven characters by pressing the return key. MODULE NAMES ARE MANDATORY ENTRIES. YOU CANNOT CONTINUE UNTIL YOU ENTER A VALID MODULE NAME.

You can now move the rectangular cursor to enter another module or:

press **e** to add an edge;
press **n** to add a node (in genus graphs only);
press the **escape key** to return to the F2Add submenu.

To return to adding modules after adding edges and/or nodes, simply press **m**. You can repeatedly cycle between adding nodes, edges, or modules by pressing the **n**, **e**, or **m** keys.

DELETING ELEMENTS FROM THE MODEL:

Select the F3Del command button by either pressing the f3 function key, clicking on the F3Del command with the left mouse button, or moving the highlight to F3Del with the cursor keys. The program will display the F3Del submemu:

Press **1** to delete an element from a genus graph.

A box cursor will appear in the lower left corner of the drawing area. Move the cursor with the cursor keys or mouse. It will move within a grid eight blocks wide by seven blocks high. To delete an element, move the box cursor to the desired location and press

return for node or click the left mouse button. The element icon, element name, and all arcs into and out of the element will be deleted.

You can now move the box cursor to delete another element or:

press **e** to delete an edge;
press **m** to delete a module;
press the **escape key** to return to the F3Del submenu.

To return to deleting nodes after deleting edges and/or modules, simply press **n**. You can repeatedly cycle between deleting nodes, edges, or modules by pressing the **n**, **e**, or **m** keys.

Press **2** to delete an edge (directed arc) from a genus graph or an undirected arc from a modular tree.

A box cursor will appear in the lower left corner of the drawing area. Move the cursor with the cursor keys or mouse. It will move within a grid eight blocks wide by seven blocks high.

To delete an edge, move the box cursor to the calling element and press the **return key**. The box cursor will change to a cross-hair cursor. Move the cross-hair cursor inside the arrow head of the arc to be deleted and press the **return key**. The arc will be deleted.

You can now move the box cursor to delete another edge or:

press **n** to delete a node;
press **m** to delete a module;
press the **escape key** to return to the F3Del submenu.

To return to deleting edges after deleting nodes and/or modules, simply press **e**. You can repeatedly cycle between deleting nodes, edges, or modules by pressing the **n**, **e**, or **m** keys.

Press **3** to delete a module from a module tree or genus graph.

A box cursor will appear in the lower left corner of the drawing area. Move the cursor with the cursor keys or mouse. It will move within a grid eight blocks wide by seven blocks high.

To delete a module, move the box cursor to the desired location and press **m** for module or click the left mouse button. The module icon, module name, and all arcs into and out of the module will be deleted.

You can now move the rectangular cursor to delete another element or:

press **e** to delete an edge;
press **n** to delete a node;
press the **escape key** to return to the F3Del submenu.

To return to deleting modules after deleting edges and/or nodes, simply press **m**. You can repeatedly cycle between deleting nodes, edges, or modules by pressing the **n**, **e**, or **m** keys.

CHANGE ELEMENT NAMES OR TYPES:

Select the F4Chg command button by either pressing the f4 function key, clicking on the F4Chg command with the left mouse button, or moving the highlight to F4Chg with the cursor keys. The program will display the F4Chg submenu:

Press **1** to change an element name on the genus graph.

A box cursor will appear in the lower left corner of the drawing area. Move the cursor with the cursor keys or mouse. It will move within a grid eight blocks wide by seven blocks high.

To change an element name, move the box cursor to the desired location and press **c** for change or click the left mouse button. The old name will be erased and the program will prompt you to enter a new element name. Names must be seven or less characters long. If you attempt to enter more than seven characters, only the first seven will be accepted and the program

will automatically continue from this point. Valid genus names must be unique and must begin with a letter. The remainder of the name may contain any combination of letters, digits or underscores. End names of less than seven characters by pressing the return key. ELEMENT NAMES ARE MANDATORY ENTRIES. YOU CANNOT CONTINUE UNTIL YOU ENTER A VALID ELEMENT NAME.

You can now move the box cursor to change another element name or press the **escape key** to return to the F4Chg submenu.

Press **2** to change an element type on the genus graph.

A box cursor will appear in the lower left corner of the drawing area. Move the cursor with the cursor keys or mouse. It will move within a grid eight blocks wide by seven blocks high.

To change an element type, move the box cursor to the desired location and press the first letter of the element type (i.e., **p**, **c**, **a**, **v**, **f**, or **t**) or click on the element icon with the left mouse button. The old icon will replaced with the new icon.

You can now move the box cursor to change another element type or press the **escape key** to return to the F4Chg submenu.

FINDING A SPECIFIC ELEMENT:

Select the F5Find command button by either pressing the f5 function key, clicking on the F5Find command with the left mouse button, or moving the highlight to F5Find with the cursor keys.

The program will prompt you to enter the name of the element you wish to find followed by the **return key**. You must enter the name exactly as it is listed in the database. The program will find this element on the genus graph and move the box cursor around it.

You can now enter another name or press the **escape key** to select another command button.

MOVING ELEMENTS OR SUBTREES:

This function is not implemented in the version of INTUITION. Selecting the F6Move command button has not effect.

LOADING A MODEL FROM DISK:

Select the F7Load command button by either pressing the f7 function key, clicking on the F7Load command with the left mouse button, or moving the highlight to F7Load with the cursor keys.

The program will prompt you to enter the filename. It will then load the file. The selected file must be on the disk in drive A. WARNING: SELECTING THIS OPTION DESTROYS ANY GRAPHS IN THE CURRENT WORKSPACE.

SAVING A MODEL TO DISK:

Select the F8Save command button by either pressing the f8 function key, clicking on the F8Save command with the left mouse button, or moving the highlight to F8Save with the cursor keys.

The program will prompt you to enter the filename. It will then save the file to the disk in drive A. Two files are created. One contains the screen image information and will have the postfix **.img** added to the name you specify. The other contains the text data needed to describe the model and will have the postfix **.dat** added to filename. After saving a file, you can continue to add to the model; however, any additions will be lost unless you resave the model.

CONNECTING TO A DATABASE MANAGEMENT SYSTEM:

Select the F9dbms command button by either pressing the f9 function key, clicking on the F9dbms command with the left mouse button, or moving the highlight to F9dbms with the cursor keys.

The program will open a connection to the ORACLE RDBMS. Pressing the key a second time will disconnect you from the database. ORACLE must be properly installed on the system you are using or the command will fail.

QUITTING THE PROGRAM:

Select the F10Quit command button by either pressing the f10 function key, clicking on the F10Quit command with the left mouse button, or moving the highlight to F10Quit with the cursor keys.

The program will ask you to confirm that you want to exit the program. Press **y** to confirm. Press **n** to continue in the program. WARNING: EXITING THE PROGRAM DESTROYS THE CURRENT MODEL UNLESS IT WAS PREVIOUSLY SAVED TO DISK.

LIST OF REFERENCES

1. Simpson, H., "A Human-Factors Style Guide for Program Design," Byte, v. 7, no. 4, pp. 108-132, April 1982.

2. Dolk, D. R., Model Management and Structured Modeling: The Role of an Information Resource Dictionary System, Working Paper No. 87-12, Naval Postgraduate School, Monterey, California, June 1987.

3. Geoffrion, A. M., "An Introduction to Structured Modeling," Management Science, v. 33, no. 5, pp. 547-588, May 1987.

4. Dolk, D. R. and Konsynski, B. R., "Model Management in Organizations," Information and Management, v. 9, no. 1, pp. 35-47, August 1985.

5. Gorry, G. A. and Krumland, R. B., "Artificial Intelligence Research and Decision Support Systems," in J. L. Bennett (ed.), Building Decision Support Systems, pp. 206-208, Addison-Wesley, 1983.

6. Geoffrion, A. M., "Structured Modeling," research monograph, Graduate School of Management, University of California at Los Aangeles, January 1985.

7. Scott, J. E., Introduction to Interactive Computer Graphics, John Wiley & Sons, Inc., 1982.

8. Bennett, J. L., "Analysis and Design of the User Interface for Decision Support Systems," in J. L. Bennett (ed.), Building Decision Support Systems, pp. 41-64, Addison-Wesley, 1983.

9. Foley, J.D., "The Human Factors--Computer Graphics Interface," In Symposium Proceedings, Human Factors and Computer Science, Washington, D.C., Potomac Chapter of the Human Factors Society, pp. 103-114, 1 June 1978.

10. Pressman, R. S., Software Engineering,A Practitioner's Approach, 2d ed., McGraw-Hill, Inc., 1987.

11. Carey, T. T. and Mason, R. E. A., "Information System Prototyping: Techniques, Tools, and Methodologies," Infor, v. 21, no. 3, pp. 177-191, August 1983.

12. Brooks, F., The Mythical Man-Month, Addison-Wesley, 1985.
13. Wyant, M. A., Design and Implementation of a Prototype Graphical User Interface for a Model Management System, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1988.

BIBLIOGRAPHY

Clemence Jr., R. D., Lexicon: A Structured Modeling System for Optimization, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1984.

Greenberg, H. J., and Maybee, J. S., (ed.), Computer-Assisted Analysis and Model Simplification, Academic Press, 1981.

Groenert Jr., F. E., A Design Analysis and Implementation of a User-Friendly Interface for the UNIX Operating System, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1984.

Shneiderman, B., Human Factors in Computer and Information Systems, Winthrop Publishers, Inc., 1980.

Shneiderman, B., Designing the User Interface: Strategies for Effective Human-Computer Interaction, Addison-Wesley, 1987.

Spear, B., How to Document Your Software, 1st ed., Tab Books, Inc., 1984.

Thomas, J. C. and Schneider, M. L., (ed.), Human Factors in Computer Systems, Ablex Publishing Corp., 1984.

Traister, R. J., Programming HALO Grahics in C, Prentice-Hall, Inc., 1985.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Director, Information Systems (OP-945) Office of Chief of Naval Operations Navy Department Washington, D. C. 20350-2000	1
4.	Computer Technology Curriculum Office, Code 37 Naval Postgraduate School Monterey, California 93943-5000	1
5.	Professor Daniel R. Dolk, Code 54DK Administrative Sciences Department Naval Postgraduate School Monterey, California 93943-5000	1
6.	Professor Gordon H. Bradley, Code 52BZ Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	1
7.	Major David D. O'Dell Headquarters, U.S. Marine Corps Code MPI-40 Washington, D.C. 20380	2

Thesis
C2443 O'Dell
c.1 The design and implementation of a visual user interface for a structured model management system.